Tampere University

Aku Pohjola

Jääkiekon vaihtojen tunnistaminen paikannustiedon avulla

Faculty of Information Technology and Communication Sciences (ITC)
Diplomityö
Huhtikuu 2019

TIIVISTELMÄ

AKU POHJOLA: Jääkiekon vaihtojen tunnistaminen paikannustiedon avulla
Tampereen yliopisto
Diplomityö, 42 sivua, 0 liitesivua
Huhtikuu 2019
Tietotekniikan koulutusohjelma
Pääaine: Audio-visuaalinen signaalinkäsittely
Tarkastajat: Apul.prof. Heikki Huttunen, Prof. Hannu-Matti Järvinen
Avainsanat: ohjelmistojen validointi, automaattinen datankäsittely, ohjelmisto arkkitehtuuri, paikantaminen

Tämän opinnäytetyön tarkoituksena oli automatisoida vaihtojen tunnistaminen osana jääkiekon tilastojen automatisointia. Tavoitteena oli toteutetun ohjelmistokomponentin tulosten validoiminen ja numeerinen vertailu. Työssä kehitettiin ohjelmistokomponentti jääkiekko-ottelun vaihtojen automaattiseen tunnistamiseen paikannustiedosta. Toteutus integroitiin osaksi laajempaa tilastointijärjestelmää, jossa sen suoritusta mitattiin.

Työssä validoitiin toteutettua ohjelmistokomponenttia mittaamalla latenssia ja arvioimalla mahdollisten virheellisten tunnistuksien määrää. Komponentin toimintaa testattiin muuttamalla parametrien arvoja ja vertailemalla muutoksen vaikutusta arvioituun virheeseen ja tilastoinnin latenssiin. Mittauksissa havaittiin joitain virheitä toteutuksessa, jotka kuitenkin ovat työlle asetettujen vaatimusten rajoissa. Vastaavasti työssä mitattu latenssi ei aiheuta merkittävää ongelmaa tilastointijärjestelmän toiminnalle.Työssä tehtyjen mittausten perustella toteutus täyttää ohjelmistokomponentin toiminnalle asetetut vaatimukset.

ABSTRACT

AKU POHJOLA: Automated ice-hockey shift detection using location data
Tampere University
Diplomityö, 42 pages, 0 Appendix pages
April 2019
Master's Degree Programme in Information Technology
Major: Audio-visual signal communication
Examiner: Assoc. Prof. Heikki Huttunen, Prof. Hannu-Matti Järvinen
Keywords: sotfware validation, automated data analysis, software architecture, positioning

The purpose of this thesis was to automate detection of shifts as part of ice-hockey statistics automation. The objective was to test and validate implemented software component. In this thesis a software component was implemented to detect shifts from ice-hockey location data. The implemented software component was integrated as part of ice hockey statistics automation system.

Software component implementation was tested for latency and approximated error. In addition, the components performance was measured for multiple values of parameters and the effect towards latency and approximate error was measured. Measurement results show that the implementation has minor shift detection errors which are within tolerance set. The measured system latency for shift detection does not cause errors statistics calculation. Concluding from the measurements, the software component meets given requirements.

SISÄLTÖ

1. Johdanto 2

2. Teoria 4

2.1 Tilakone ja hystereesi 4

2.2 Kolmiopaikantaminen 5

2.3 Vuoarkkitehtuuri 7

2.4 Mikropalveluarkkitehtuuri 8

2.5 Ohjelmistotestaus 9

3. Tekninen tausta 11

3.1 Bluetooth 11

3.2 .NET Core ja C# 12

3.3 Dataflow 13

3.4 Etäkutsuprotokolla gRPC 14

4. Toteutus 16

4.1 Vaatimukset ja haasteet 16

4.2 Peli- ja paikannustiedon kerääminen 18

4.3 Ottelutiedon käsittelyn arkkitehtuuri 20

4.4 Paikannusvirheen kompensointi 23

4.5 Vaihtojen parittaminen 26

4.6 Testaaminen 28

5. Tulokset 31

5.1 Mittaussunnitelma 31

5.2 Mittaustulokset 32

5.3 Tulosten yhteenveto 37

5.4 Jatkokehitysehdotuksia 39

6. Yhteenveto 41

Lähteet 43

LYHENTEET JA MERKINNÄT

AOA	Angle of Arrival
API	Application Programming Interface
CLR	Common Language Runtime
GPS	Global Positioning System
gRPC	gRPC Remote Procedure Call
HTTP	Hyper Text Transfer Protocol
IL	Immediate Language
QPE	Quuppa positiong engine
RPC	Remote Procedure Call
REST	Representational State Transfer
SS	Signal Strength
TDOA	Time difference of Arrival
TOF	Time of Flight
TOA	Time of Arrival

1. JOHDANTO

Jääkiekko on ammattiurheilun mahdollistava urheilulaji, johon liittyy paljon harrastus- ja liiketoimintaa. Lajina jääkiekko on suomessa suosittu laji sekä harrastaa, että seurata ja saa siksi osakseen paljon huomiota. Lajista kiinnostuneet voivat seurata esimerkiksi oman suosikkijoukkueen menestymistä tai suosikkipelaajansa kehitystä. Yksi muoto kehityksen seurannalle on pelaajan tai joukkueen tilastojen seuranta. Kuitenkin teknologisista mahdollisuuksista huolimatta tilastointi on manuaalisen työn varassa.

Jääkiekon automaattinen tilastointi avaa monia uusia mahdollisuuksia lajista kiinnostuneelle. Esimerkiksi valmentajat saavat käyttönsä huomattavasti tarkemman tiedon pelaajasta. Muita kiinnostuneita pääsidosryhmiä ovat aktiiviset joukkueen kannattajat, tuomarit ja vedonlyöntitoimijat. Toteuttaakseen jokaisen sidosryhmän vaatimuksen järjestelmän täytyy toimia reaaliaikaisesti ja ilman huomattavia virheitä. Vaatimus reaaliaikaisuudesta on korostunut vedonlyönnin kannalta tilannetta miettiessä.

Selkein hyöty automaattisesta tilastoinnista on valmentajille. Tilastoimalla esimerkiksi pelaajien pulssi pelin aikana antaa aikasemmin tilastoimatonta numeerista ja vertailtavaa tietoa pelaajista. Vastaavasti tietämällä pelaajien paikat saadaan tarkasteluun pelaajan yleinen käyttäytyminen suhteessa pelaajan rooliin. Ottelunaikaiseen analyysiin vaatimuksena tulee reaaliaikaisuus ja ottelun jälkeiseen analyysin valmentaja hyötyy tilastojen kattavuudesta ja tarkkuudesta.

Vedonlyönti-järjestöille automaattinen tilastointi avaa useita vaihtoehtoa, sillä ihmisten keräämät tilastot ovat usein virheellisiä. Konellinen tilastointi poistaa myös mahdollisuuden vahingollisiin virhesyötteisiin käyttöliittymän kautta. Hyödyntääkseen automaattisen tilastoinnin saamia mahdollisuuksia tilastoja pitäisi hyödyntää reaaliaikaisesti ja tilastojen oikeellisuus on tärkeää.

Pelaajalle ja joukkueen kannattajille tilastointi tuo uusia ominaisuuksia, joita seurata ja vertailla. Pelaajien kannalta tilastojen kehittymisen seuranta tuottaa helposti seurattavia ja verrattavia tietoja kuten laukaisujen nopeudet, luistelunopeus tai

jäälläoloaika. Vastaavasti joukkueetta ja pelaajia kannattavat ihmiset voivat seurata pelaajien kehitystä.

Jääkiekon automaattinen tilastointi koostuu useista tekijöistä. Tämän työn tarkoituksena on automatisoida vaihtojen tunnistaminen, joka on yksi olennainen osa jääkiekon tilastojen automatisointia. Työssä tullaan myös kehittään ohjelmistokomponentti, joka tunnistaa jääkiekon pelaajien vaihdot käyttäen paikannustietoa. Työn tavoitteisiin kuuluu myös toteutetun ohjelmistokomponentin tulosten validoiminen ja numeerinen vertailu.

Työ rajataan käsittelemään vain vaihtojen tunnistamiseen ja tilastointiin vaikuttavia tekijöitä. Työssä ei oteta kantaa kehitysympäristön valintaan tai ohjelmiston suorittamiseen tuotantoympäristössä. Työssä esitellään kuitenkin yleiskuvaa tilastointijärjestelmän toimintaympäristöstä.

Luvussa 2 esitellään teoriaa toteutuksen taustalla. Luvussa 3 käydään läpi teknistä taustaa tehtävälle ohjelmistokomponentille. Lukuun 4 siirryttäessä työssä käydään läpi toteutusta ja yleistä kuvaa järjestelmästä. Työn mittauksia ja mittausten perustella tehtyjä yhteenvetoja käydään läpi luvussa 5. Lopuksi luvussa 6 esitellään yhteenveto koko työstä.

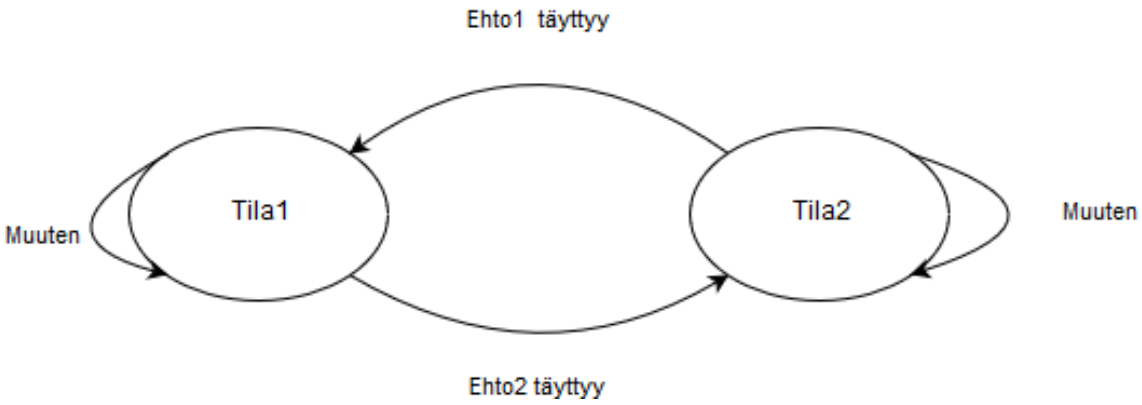
2. TEORIA

Tässä luvussa esitellään yleistä tietoa opinnäytetyössä hyödynnettävistä tilakoneesta ja hystereesistä, ohjelmistokehityksen näkökulmasta. Samalla osissa esitellään tietoa kolmiopaikannuksen tyypeistä. Kolmiopaikannuksen jälkeen luvussa siirrytään käsittelemään ohjelmisto arkkitehtuureja, joista esitellään mikropalveluarkkitehtuuri ja vuoarkkitehtuuri. Viimeisenä asiana luvussa käsitellään ohjelmistojen testaamista.

2.1 Tilakone ja hystereesi

Tilakone sisältää tilamuuttujia, jotka määrittävät tilakoneen tilaa sekä komentoja, jotka muuttavat järjestelmän tilaa [23], mallintaen järjestelmän tilaa siirtymällä tilojen väleillä määriteltyjen ehtojen mukaisesti. Tilojen äärellisyyden mukaan tilakoneita voidaan kategorisoida kahteen eri luokaan. Tilojen määrän ollessa äärellinen tilakonetta kutsutaan äärelliseksi automaatiksi. Vastaavasti tilojen määrän ollessa ääretön tilakonetta kutsutaan äärettömäksi automaatiksi.

Kuvassa 2.1 esitetään yksikertainen esimerkki kaksivaiheisesta tilakoneesta. Kuvassa 2.1 esitetyssä tilakoneessa siirytään *tilasta 1 tilaan 2* vain, jos tilakoneen esittämä *ehto2* osoittuu todeksi. Ehto voi olla esimerkiksi kynnysarvon ylitys tai alitus. Esimerkiksi yksinkertainen luokittelija, joka jakaa näytearvoja kahteen luokaan



Kuva 2.1 *Kaksivaiheinen tilakone tilakaaviona*

Arvot	1	0,76	0,51	0,26	0,1	0,28	0,55	0,49	0,51	0,34
Symmetrinen ehto	B	B	B	A	A	A	B	A	B	A
Asymmetrinen ehto	B	B	B	B	A	A	A	A	A	A

Taulukko 2.1 *Esimerkki hystereesi ja tilakoneen erosta*

kynnysarvon perusteella.

Hystereesi on äärillinen automaatti, joka muistaa edelliset tilansa [15]. Hystereesi on tilakoneen eritystapaus, jonka tavoitteena on muuttaa jatkuvan funktio tai ilmiö diskreetiksi arvoksi siten, että muutospisteen lähellä ei ole värähtelyä. Muunnospisteen ympärillä olevaa värähtelyä kompensoidaan tekemällä muunnosehdoista epäsymmetriset.

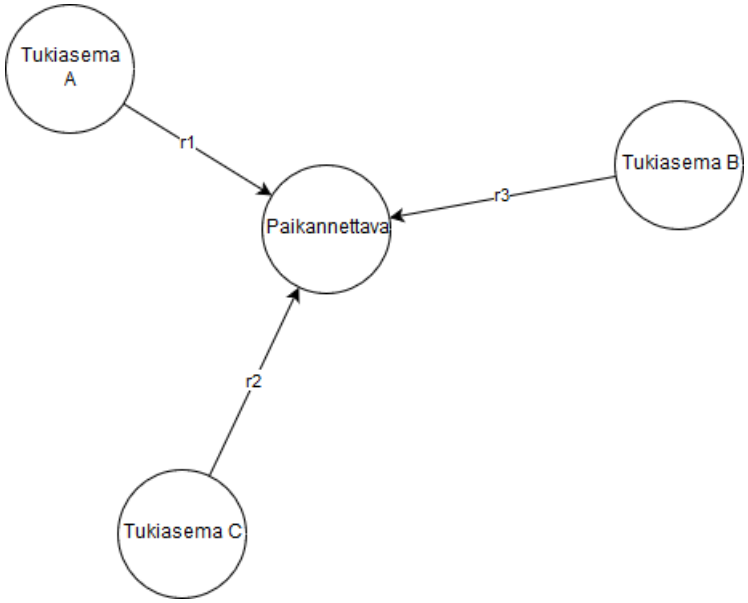
Taulukossa 2.1 esitetään yksinkertainen esimerkki hystereesin ja tilakoneen välillä. Tilakoneen tapauksessa luokittelu perustui vain yksittäiseen pisteeseen siten, että alle 0,5 olevat arvot olevat luokkaa A ja muut arvot luokkaa B. Verrattuna hystereesi hyödyntää aikaisempia näytteitä oman tilansa määrittämiseen. Hysteerisissä luokitellaan edelliseen tilaan, ellei ylitetä tai aliteta kynnys arvoja 0,25 ja 0,75 eli luokkaan B luokitellaan, kunnes saadaan syöte arvoltaan alle 0,25, vastaavassa luokkaan A luokitellaan, kunnes saadaan näytepiste, joka on yli 0,75.

Taulukossa 2.1 on havaittavissa eroa asymmetrisen ja symmetrisen tilasiirtymän välillä. Tilasiirtymäehtojen välinen ero on havaittavissa esimerkin lopussa olevassa muunnosarvon ympärillä tapahtuvassa värähtelyssä. Symmetriseen siirtymään perustuva tilakone luokittelee kynnysarvon ympäriltä arvot omiin luokkiinsa aiheuttaen värähtelyä luokittelussa. Verrattuna asymmetriseen siirtymään perustuva tilakone luokitteli kaikki arvot samaan luokkaan. Mikäli värähtely on kohinaa, niin hystereesi toimii kohinasietoisemmin verrattuna symmetrisestä siirtymäehtoa käyttävään tilakoneeseen.

2.2 Kolmiopaikantaminen

Kolmiopaikannuksessa tavoitteena on paikantaa signaalia lähettävä kohde. Kolme yleisintä menetelmää perustuvat joko signaalin vahvuuteen, signaalin saapumisaikaan tai kulmamittaukseen [11]. Vaihtoehtoisesti paikantaminen voidaaan tehdä käyttämällä Global Positioning System:iä (GPS), joka paikantaa pisteitä satelliiteilla [24], joka itssesään perustuu kolmiomittaamiseen. GPS-paikannuksen tarkkuus kuluttajalle on noin 10 metriä [24, 10].

Signaalin vahvuuteen perustuvissa menetelmissä eli Signal Strength (SS) menetel-

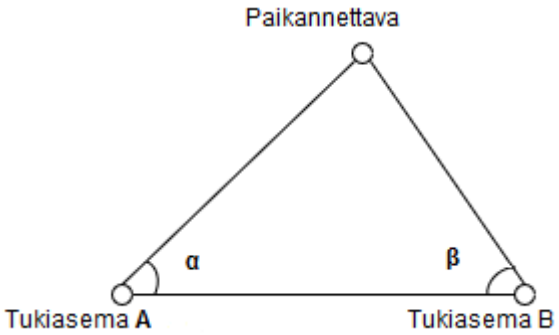


Kuva 2.2 *Paikantaminen käyttäen kolmea tukiasemaa*

missä mitataan signaalin vahvuus kolmella tai useammalla tukiasemalla. Saatujen mittausten perusteella pystytään arviomaan jokaisen tukiaseman etäisyys signaalin lähteeseen suhteessa toisiinsa. [10, 11] Etäisyyden arviontiin voidaan käyttää myös signaalin vastaaonottoajan mittausta eli Time of Flight (TOF)- menetelmää. TOF-menetelmät voidaan jakaa Time of Arrival (TOA) - mittauksiin ja Time difference of Arrival (TDOA). TOA-mittauksissa lähettäjä lähettää signaalin ajanhetkellä, jonka vastaanottaja tietää. Vertaamalla vastaanoton ajanhetkeä signaalin oletettuun lähetyshetkeen, saadaan arvioitua vastaanottajan ja lähettäjän välinen etäisyys. [10]. Vaihtoehtoinen menetelmä on verrata vastaanottajien vastaanottoaikoja keskenään [11], jota kutsutaan TDOA:ksi.

Kuvassa 2.2 esitetään esimerkki kuinka sekä signaalin vahvuuteen, että signaalin vastaanottoaikaan perustuvissa menetelmissä voidaan paikantaa piste. Kuvan 2.2 esittämässä tapauksessa tiedetään tarkasti tukiasemien sijainnit halutussa koordinaatistossa. Luontaisesti tukiasemien sijainnit toisiinsa nähden ovat tiedossa. Jokainen tukiasema tällöin arvioi etäisyyttä pisteeseen halutulla menetelmällä. Tietämällä vähintään suhteellinen etäisyys tukiasemien ja paikannettavan pisteen välillä, voidaan piste paikantaa.

Paikantamiseen voidaan käyttää Angle of Arrival (AOA) eli kulmamittausta. Kulmamittauksessa käytetään vähintään kahta tukiasemaa, joiden sijainnit tiedetään etukäteen. Mittaamalla vastaanotettavan signaalin kulma saadaan määritettyä pisteen sijainti tasossa. [11, 10] Etuna kulmamittauksessa on puute alle mikrosekunnin tarkkuukuusvaatimuksesta ajan mittauksissa tai vaatimus synkronisaatiosta, vas-



Kuva 2.3 *Esimerkki kulmamittauksesta*

taavasti ongelmana kulmamittauksessa on laajempi vaatimus kalibraatiolle[11].

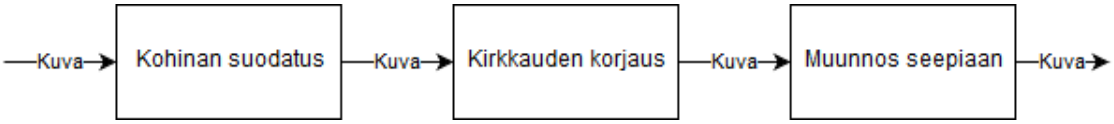
Kuvassa 2.3 esitetään esimerkki kulmamittaamisesta. Esimerkissä mitataan kulmat α ja β. Kuvaan on selkeyden vuoksi merkitty pisteiden sijainnit kaksiulotteisessa koordinaatistossa. Signaalin vastaanottokulmien α ja β perusteella voidaan laskea paikannettava piste, kun tiedetään tukiasemien sijainti toisiinsa nähden.

2.3 Vuoarkkitehtuuri

Vuoarkkitehtuurissa on komponentteja, jotka ottavat joukon sisääntuloja muuttaen ne joukoksi ulostuloja. Muunnos tapahtuu yleensä paloittain, jolloin ulostuloja voidaan laskea samanaikaisesti kuin sisääntulo syntyy. [7] Komponentteja jotka käsittelevät dataa kutsutaan suodattimiksi ja komponettejä jotka välittävät syötteitä kutsutaan putkiksi [7, 22]. Syötteitä välittävien komponenttien eli putkien kanssa on tärkeää, että putket keskittyvät datan välittämiseen ja eivät muuta syötteitä lainkaan [22].

Esimerkkinä vuoarkkitehtuurista voidaan käyttää kääntäjää, sillä kääntäjää voidaan mallintaa liukuhihnana [21, 7]. Kääntäjässä jokainen vaihe ottaa sisäänsä edellisen vaiheen tuottamuksen ja tekee sille muunnoksen välittäen sen eteenpäin. Toteutus on tällöin vuoarkkitehtuurin mukainen, vaikka se ei välttämättä ole rinnakkainen. [7]

Vuoarkkitehtuuri sallii suunnittelijan käsitellä järjestelmää yksittäisten suodattimien koostumuksena. Toisena vuoarkkitehtuurin etuna on komponenttien uudelleenkäyttömahdollisuus, sillä komponentteja voidaan kiinnittää vapaasti toisiinsa, kunhan niiden tulos- ja syöttöformattit ovat samat. Samalla vuoarkkitehtuurin etuna on komponenttien helppo korvattavuus ja vastaavasti uusien suodattimien lisäys järjestelmään on helppoa. [7]



Kuva 2.4 *Esimerkki vuo arkkitehtuurista*

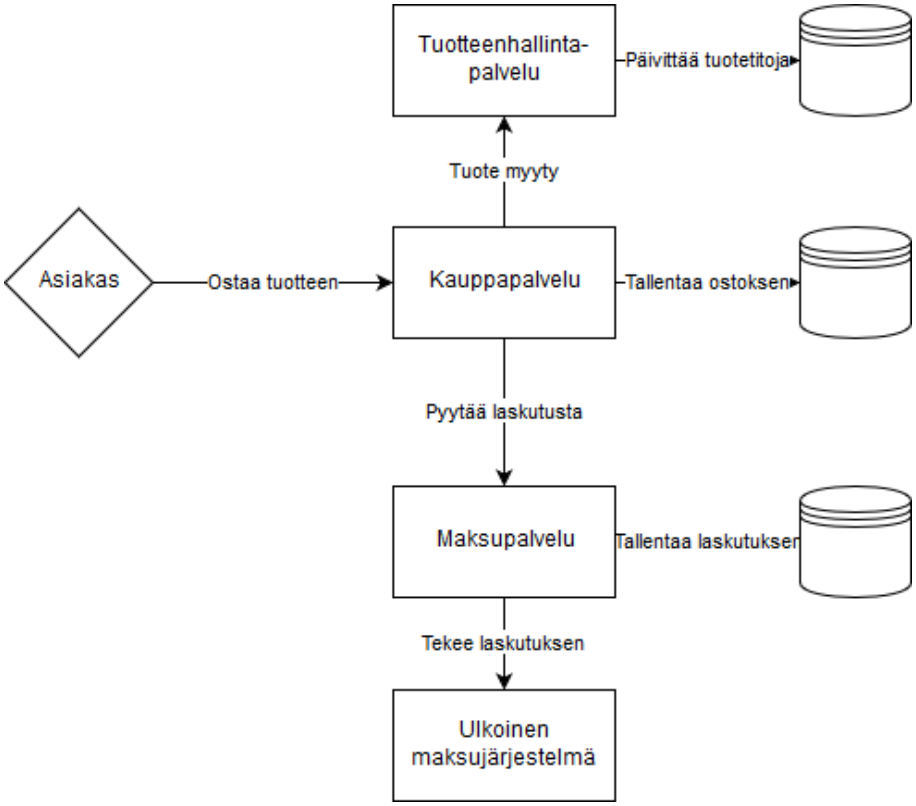
Kuvassa 2.4 esitetään yksinkertaistu arkkitehtuuri kuvankäsittelyn järjestelmästä. Kuvassa esitetään kuvan muokkaamista vaiheittain, esimerkiksi kuvalle tehdään kohinan suodatusta, jonka jälkeen kuvalle tehdään kirkkauden korjaus ja lopulta muunnetaan kuva seepia väreihin. Huomioitavaa on, että jokaisessa välivaiheessa tuotos on valmis kuva. Vastaavasti asettamalla jokaiselle suodattimelle sama rajapinta, ne tuottavat ja ottavat sisään identtisiä alkioita. Symmetrisyys sisään- ja ulostulossa mahdollistaa suodattimien lisäämisen, poistamisen ja siirtämisen vapaasti.

2.4 Mikropalveluarkkitehtuuri

Mikropalveluarkkitehtuurissa mikropalvelut ovat pieniä autonomisia palveluja, jotka viestivät keskenään. Mikropalveluarkkitehtuurin ideana on jakaa kokonaisuus useaan pienempään kokonaisuuteen, joilla on yksi selkeä tehtävä. Vastaavasti jokainen palvelu on itsenäinen komponentti, joka tarjoaa ohjelmistorajapinnan verkkokutsujen yli [20]

Mikropalveluarkkitehtuurin etuja ovat esimerkiksi skaalattavuuden helppous, sillä uusia instansseja palvelusta voidaan luoda rinkkakkain. Samalla mikropalvelut voivat olla helpompia asentaaa, sekä ohjlemistot ova vähemmän riippuvaisia toisistaan ja virheiden alueet ovat pienmpiä .[1] Muita etuja mikropalveluarkkitehtuurilla on toteutusteknologian vapautuneisuus, sillä jokainen palvelu voidaan toteuttaa eri teknologialla mahdollistaen oikean teknologian valinnan ongelman ratkaisemiseen [20]. Huomattava etu mikropalveluarkkitehtuurilla on toteutuksen korvattavuus, sillä yksittäisiä pieniä palveluja on huomattavasti helpompaa poistaa käytöstä tai korvata uudella toteutuksella tulevaisuudessa [20].

Kuvassa 2.5 esitetään esimerkki verkkokaupan toteuttamisesta mikropalveluarkkitehtuurilla. Toteutuksessa jokainen palvelu voi esimerkiksi tarjota Hyper-Text Transfer Protocol (HTTP) Representational State Transfer (REST) Appliation Programming Interfacen (API). Tarjoamalla HTTP REST API:n palvelut voivat olla toteutettuja vapaasti haluamallaan teknologialla ja voivat vaihtaa käytettyä teknologiaa vapaasti. Huomattavaa esimerkissä on myös, kuinka jokaisella palvelulla on oma tietokantansa, tällöin palvelut eivät ole sidottuja toisiinsa tietokannoistansa [20].



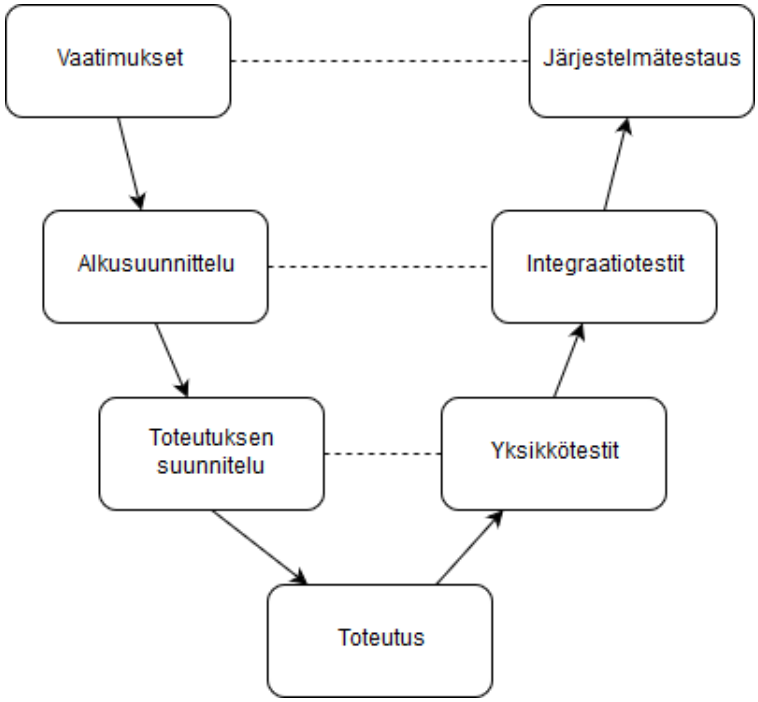
Kuva 2.5 *Verkkokauppan mikropalveluarkkitehtuuri*

2.5 Ohjelmistotestaus

Ohjelmistojen testaus on periaatteellisesti kiinnostunut järjestelmän käyttäytymisestä eikä järjestelmän toteutuksesta [12, p. 5]. Testausta voidaan jakaa toiminnalliseen testaukseen, missä toteutusta testataan puhtaasti sen sisään- ja ulostulojen perusteella ilman tietoa toteutuksesta. Kontrastissa ohjelmistoja voidaan myös testata koodipohjaisella testauksella, jolloin testauksessa on tiedossa toteutus yksityiskohdat ja testejä voidaan suunnitella toteutuksen perusteella. [12, .p 6-9]

Kuvassa 2.6 esitetään testauksen v-malli. Kuvassa sidotaan suunnittelun vaiheet vaatimusmäärittely, esisuunnittelu ja toteutuksen suunnittelu testauksen tasoihin järjestelmätestaus, integraatiotestaus ja yksikkötestaus [12, p.208]. V-malli kuvaa ohjelmiston testaamisen tasoja nousten yksikkö testaamisesta järjestelmä testaamiseen. Testaamisen parhaiten ymmärretty taso esitetyistä tasoista on yksikkötestaaminen ja siinä testaan yksittäistä komponenttia. Integraatiotestaamisen tavoitteena on yhdistää yksittäisiä komponentteja laajemmiksi kokonaisuusiksi ja järjestelmätestaus lähenee asiakaan hyväksymistestausta. [12, .p 208].

Yksikkötestauksessa testataan yksittäistä komponenttia ohjelmistossa. Testauksen menetelmä voi esimerkiksi olla raja-arvoihin perustuva testaus. Yleisessä raja-arvoihin



Kuva 2.6 *Testauksen V-malli [12]*

perustuvassa testauksessa funktiota testataan sen käsittelemien arvojen rajoilla[12, .p79 81]. Raja-arvotestauksen lisäksi, yksikkötestausta voidaan toteuttaa funktionaalisen testauksen muotona ekvivalenssitestauksella, jossa muuttujia luokitellaan sopivuuden mukaan ja tällöin luodaan ekvivalenssiluokkia testisyötteille, joita testataan toteutettua komponenttia vastaan [12, .p 99-101].

Integraatiotestaus on heikoiten ymmärretty ja siten yleisesti heikoiten toteutettu testauksen taso esitetyistä testauksen tasoista [12, p.229]. Integraaatio testauksessa olennaista on valmiin komponentin testaaminen, mutta kuitenkin rajoittaen testauksen mittaluokaa yksittäisten komponettien keskinnäisen toiminnan testaamiseen. Testauksesa voidaan korvata muita sivukomponentteja esimerkiksi tyngillä pienentääkseen pinta-alaa, josta virheitä voi syntyä. Kun testattut komponentit toimivat keskenän virheettömästi, voidaan testattavia komponenttejä lisätä korvaamalla tynkiä.Vaihtoehtoisesti testaamista voidaan toteuttaa funktiokutsu-tason integraatiolla, jossa kutsuttuja funktiota korvataan yksi kerrallaan oikealla toteutuksella. [12, p.229-240]

Järjestelmätestauksessa tarkoite ei ole löytää vikoja vaan osoittaa järjestelmän oikea toimivuus. Tavoitteen takia järjestelmätestauksessa edetään vaatimusmäärittelypohjaisesti verrattuna ohjelmakoodi-pohjaiseen testaamiseen. Yksikkönä järjestelmätestaamisessa voidaan käyttää säiettä, joka järjestelmätestauksen konseptissa on esimerkiksi yleinen käyttötapaus tai järjestelmätason testitapaus. [12, .p 253-254]

3. TEKNINEN TAUSTA

Luvussa esitellään teknistä taustaa työn toteutukselle. Ensimmäisessä osiossa esitellään Bluetooth-standardia ja sen käyttöä paikannukseen. Seuraavassa osiossa esitellään ohjelmistokehitysalustaa .NET Core ja ohjelmointikieltä C#, sekä niiden rinnakkaisuuden toimintaa. Kolmannessa osiossa esitellään työssä käytettyä ohjelmointikirjastoa Dataflow. Viimeisessä osiossa esitellään toteutuksessa käytetty etäkutsuprotokolla.

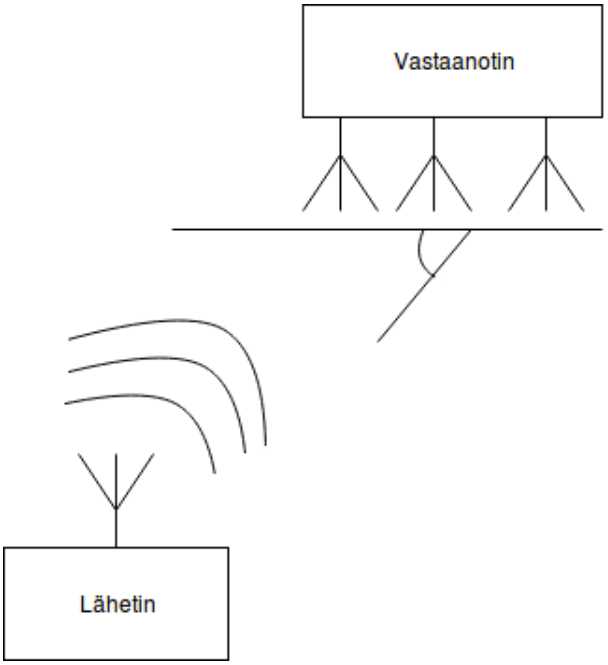
3.1 Bluetooth

Bluetooth on standardi lyhyen kantaman langattomalle viestinnälle käyttäen radioteknologiaa [14]. Bluetoothia käytetään korvaamaan esimerkiksi tietokoneiden oheislaitteiden, kuten hiirien, johdotukset [13]. Muita yleisiä käyttökohteita ovat esimerkiksi sulautetut järjestelmät [14].

Teknisiltä ominaisuuksiltaan Bluetooth mahdollistaa asynkronisen datan siirron tai synkronisen audiovuon siirtämisen.[3] Tiedonsiirtoon Bluetooth käyttää radioteknologiaa toimien 2.4 GHz taajuusalueella. Bluetooth on Bluetooth Special Interest Group:in kehittämä ja ylläpitämä standardi.

Bluetoothia voidaan myös käyttää sisätilapaikantamiseen käyttämällä kolmiopaikannusta. Standardina Bluetooth soveltuu reaaliaikaisiin paikannusjärjestelmiin. Bluetoohia käyttävä suunnanetsintä tukee kulmamittauspohjaista AoA-menetelmää. Menetelmä perustuu lähettävään komponenttiin ja joukkoon vastaanottavia antenneja, joiden kulmamittauksen perustella suunta-arvio tehdään. [4]. Paikannussovelluksessa Bluetooth tarjoaa energiatehokaamman ratkaisun verratuna esimerkiksi WiFi toteutettuihin sovelluksiin [5].

Kuvassa 3.1 esitelään [4] mukainen esimerkki AoA -menetelmän mukaisesta mittauksesta. Kuvassa 3.1 on näkyvissä yksi lähetinkomponentti ja useampi vastaanotinkomponentti. Vastaottimet arvioivat lähettimen sijaintia perustuen vastaanottimien välisiin etäisyyksiin [4]



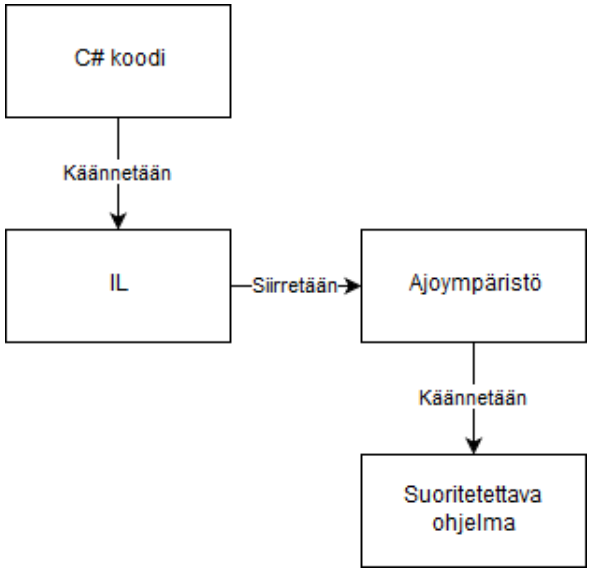
Kuva 3.1 *Kulmamittaus käyttäen Bluetoothia*

3.2 .NET Core ja C#

Ohjelmistokehitysalusta .NET Core toimii useassa käyttöjärjestelmässä. Ohjelmaa suoritetaan käyttäen Common Language Runtimea (CLR), joka muokkaa kääntäjän tuottamaa Immediate Language (IL) -koodia suoritettavaan natiivin koodiin. [16] Ohjelmointikieli C# on ohjelmointikieli, joka on käännettävissä .NET Coren käsittelemään IL-koodiin. Paradigmaltaan C# on olio-ohjelmointikieli, joka tukee myös komponentti-orientoitua ohjelmointia [19]. Muistinhallinta CLR:lle kääntyvissä kielissä on toteutettu roskien keruulla eli tarkastelemalla ajoaikaisesti viitteiden määrää muistialkioihin jolloin ohjelmoijan ei tarvitse hallinnoida muistia käsin.

Kuvassa 3.2 esitetään yksinkertaistettu näkymä, kuinka ohjelmointikieli C# saatetaan suoritukseen. Ensin ohjelmakoodi käännetään IL-koodiksi. Käännetty IL-koodi siirretään suorittavaan ympäristöön, jossa ajoympäristössä oleva CLR ottaa IL-koodin suoritukseen. Lopulta CRL-kääntää IL-koodin ajoaikaisesti natiiviksi suoritettavaksi ohjelmaksi.

Ajoympäristö CRL tukee rinnakkaista suoritusta varten useita rakenteita. Tuettuja rakenteita ovat esimerkiksi säikeet, prosessit ja tehtävät. Erot rinnakkaisuus rakenteiden välillä ovat selkeät. Esimerkkisi prosessi suorittaa itsensä täysin eristetysti omassa muistiavaruudessaan. Vastaavasti säikeet suorittavat ohjelmaa prosessin sisäisessä jaetussa muistiavaruudessa. Eroavana rakenteena ovat tehtävät, jotka ovat abstraktio rinnakkaisuudesta, joka on rakennettu säikeen päälle.[18].



Kuva 3.2 *C#:n kääntäminen ohjelmakoodiksi*

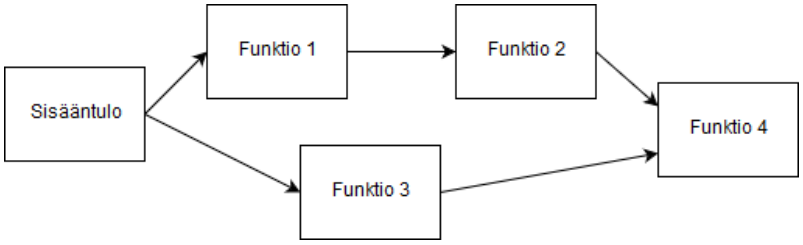
Tehtävät ovat .NET kielissä kevyempi rakenne kuin käyttöjärjestelmän säikeet ja mahdollistavat tarkemman käsittelyn ohjelmallisesti. Molempien mainittujen syiden takia .NET ympäristössä Task Parallel Library on suositeltu tapa käsitellä multisäikeistä, asynkronista ja rinnakkaista koodia. [18] Sisäisesti tehtävät käyttävät suoritukseensa säijejoukkoa, jolloin tehtäviä voidaan suorittaa rinnakkaisesti, mutta samalla jokaista tehtävää varten ei tarvitse käynnistää uutta säijettä, mahdollistaen hienojakoisen rinnakkaisuuden.

Esimerkkinä tehtävien käyttämisestä asynkroniseen ohjelmointiin voidaan käyttää kommunikointia ulkoisten palvelimien kanssa. Tekemällä esimerkiksi HTTP pyynnön palvelimelle voidaan käynnistää tehtävä, joka suoritetaan, kun HTTP-kutsu palauttaa arvon. Tällöin voidaan käyttää aika, joka kyselyn lähettämisen ja vastaanottamisen välillä kuluu, tehden muuta laskentaa ohjelmistossa. Esimerkiksi verkkopalvelin, joka kommunikoi tietokannan kanssa, kuluttaa yksittäisestä kyselystä huomattavan osan aikaa odottaen vastausta tietokannalta, voi käyttää tehtäviä vähentääkseen suorittimen odottamiseen kuluttamaa aikaa.

3.3 Dataflow

Dataflow on ohjelmistokirjasto rinnakkaisohjelmoinnin helpottamiseen, joka on toteutettu sisäisesti C#:lla [17]. Ohjelmistokirjasto mallintaa vuoarkkitehtuurimallia ja arkkitehtuurimallin mukaisesti kirjasto tarjoaa abstraktion dataa käsitteleville komponenteille ja niiden kiinnittämiseen toisiinsa. Dataflow-kirjasto soveltuukin hyvin sovelluksiin joilla on tarve käsitellä dataa yhtä nopeasti, kuin tietoa tuotetaan [17].

Kuva 3.3 *Esimerkki lihkuhihnasta dataflow kirjastolla*



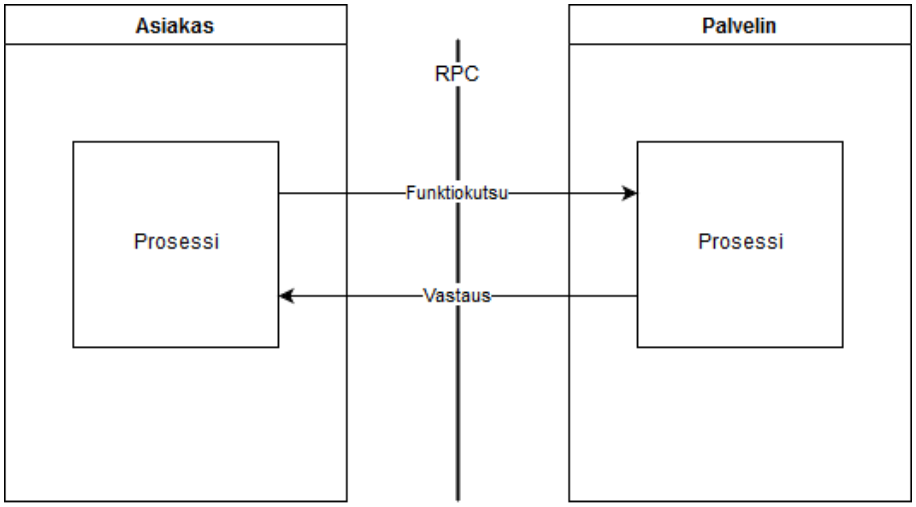
Kuvassa 3.3 esitetään yksinkertainen vuoarkkitehtuurin mukainen malli. Kirjaston tarjoamana etuna jokaista funktiota voidaan suorittaa rinnakkaisesti. Lohkojen rinnakkainen suoritus mahdollistaa liukuhihnamaisen suorituksen. Esimerkiksi kuvan 3.3 funktio 1 voi käsitellä jo seuraavaa datapistettä ja syöttää funktion 2 puskuriin käsiteltäväksi. Vastaavasti kirjasto tarjoaa rajapinnan syötteiden synkronoinnille. Kuvassa 3.3 funktioden 2 ja 3 syötteet voidaan synkronoida funktiolle 4 syötettäessä siten, että funktio 4 saa syötteet aina monikkona.

Sisäisesti Dataflow käyttää C#:n tehtäviä. Tehtävien käyttäminen mahdollistaa kevyen rinnakkaisuuden, sillä jokainen lohko voidaan abstrahoida tehtäväksi antaen kielen sisäisen vuorontajan määrittää suoritusjärjestyksen ja rinnakkaisuuden tason. Kirjasto tarjoaa myös rinnakkaisuutta yksittäisen lohkon suoritukseen. Esimerkiksi mikäli esimerkikissä 3.3 funktio 1 on pullonkaula suorituksessa ja sen syötteet eivät ole toisistaan riippuvaisia, voidaan lohkoa rinnakkaista kopioimalla lohkoa. Tällöin voidaan suorittaa rinnakkaisesti funktion 1 kutsuja.

3.4 Etäkutsuprotokolla gRPC

Remote Procedure Call (RPC) on teknologia, joka on suunniteltu sallimaan etäprosessin käyttämisen yhtä helpoksi kuin paikallisten prosessien [2]. RPC-toteutukset piilottavat yksityiskohdat hajauttamisesta kuten enkoodauksen ja dekoodauksen. Vastaavasti RPC ei vaadi ohjelmoijalta tietoa hajautetun prosessin toteutuksesta, sillä RPC piilottaa esimerkiksi etäpalvelimen yksityiskohdat toteutuksesta. [8]

Kuvassa 3.4 esitetään esimerkki RPC:n käyttämisestä kommunikointiin. Esimerkissä asiakas kutsuu palvelimen tarjoamaa funktiorajapintaa yli RPC:n. Asiakkaan ei tarvitse tietää palvelimen toteutuskieltä tai kuinka RPC-kirjasto siirtää funktiokutsun parametrit. Lisäksi etuna RPC:ssä on tuki ohjelmiston muutokselle, kunhan tarjottu rajapinta pysyy samana [8]. Etäkommunikointiin suunniteltu gRPC Remote Procedure Call (gRPC) on RPC-järjestelmä, joka käyttää kommunikointiin protokolana HTTP 2:sta. gRPC:ssä on palvelinkomponentti, johon asiakkaat ottavat yhteyden käyttäen ohjelmointikielelle luotua kirjastoa. Totutettuja ohjelmointikieliä



Kuva 3.4 *Asikaan ja palvelimen kommunikatio RPC:llä*

ovat esimerkiksi C#, C++ ja Go. [9]

Sisäisesti gRPC käyttää protobuf3:sta palvelujen rajapintojen määrittelyyn. Protobuf kuvauksesta käännetään ohjelmointikielelle kirjastot sekä palvelun tarjoamiseen, että palveluun yhteydenottoon [9]. Tarjoamalla rajapintakuvauksen gRPC on Interface Defination Language ja mahdollistaa yhtenäisen kutsumisen usean ohjelmointikielen välillä. [8]

4. TOTEUTUS

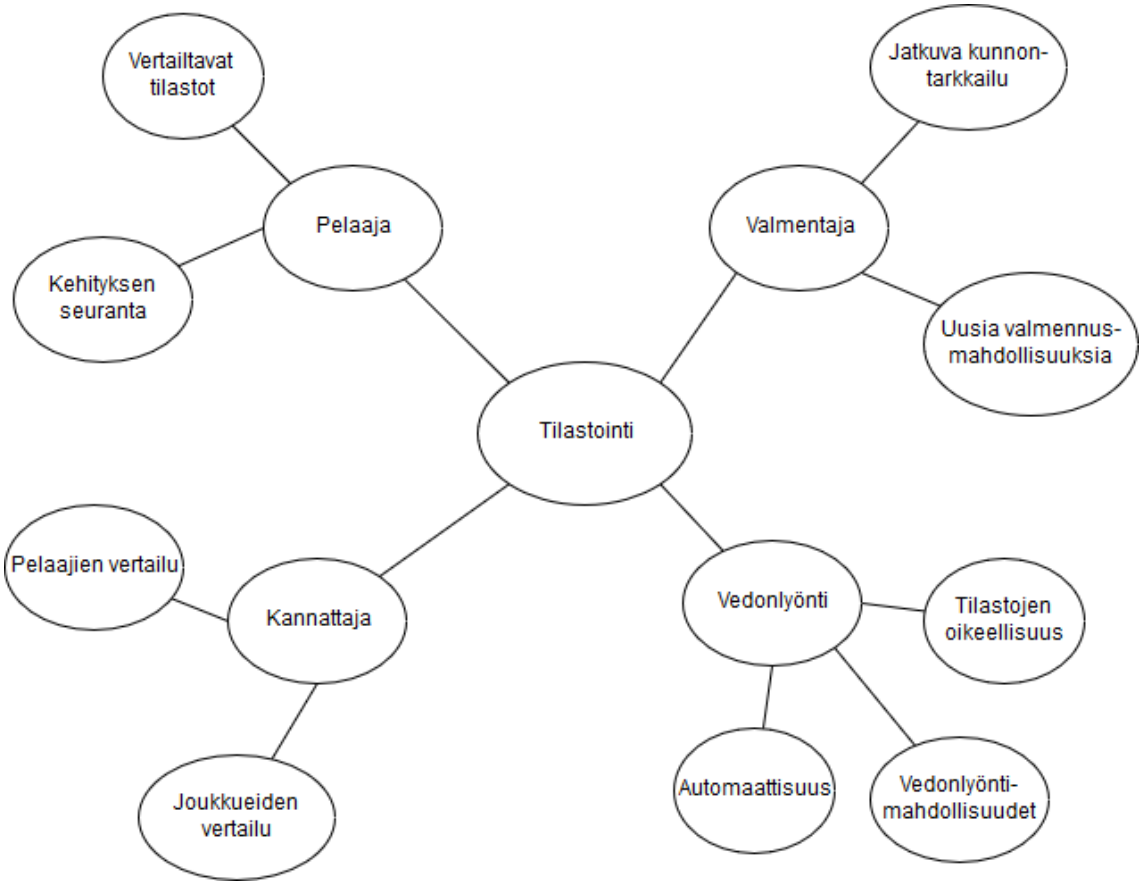
Tässä luvussa käsitellään järjestelmän taustoitusta ja toimintaympäristön vaatimuksia. Vaatimusten jälkeen esitetään yleiskatsaus ottelutiedon keräämiseen. Ottelutiedon keräämisen jälkeen siirrytään kuvaamaan olennaisia järjestelmän komponentteja otteludatan käsittelyä varten. Luvussa siirrytään arkkitehtuurikuvauksen jälkeen käsittelemään paikannusvirheen kompensointia. Virheen kompensoinnin jälkeen kuvataan vaihtojen parittamista ja parittamiseen liittyviä ongelmia. Viimeisessä osiossa käsitellään järjestelmän testaamista.

4.1 Vaatimukset ja haasteet

Jääkiekon peliajan tilastointi käsin on työlästä ja tilastointi sisältää usein virheitä, samalla tilastoinnin haasteena on se, että tulokset eivät ole saatavilla reaaliaikaisesti. Jääkiekon peliajalla tarkoitetaan tässä työssä pelaajan aikaa jäällä. Jääkiekon pelaajien vaihtojen automaattinen tilastointi ratkaisee ongelman esimerkiksi pelaajakohtaisen peliaikaseurannan oikeellisuudesta, luotettavuudesta ja reaaliaikaisuudesta. Tästä johtuen esimerkiksi inhimillisen virheen mahdollisuus pelaajien vaihtojen seurannassa pienenee, sillä automaattinen järjestelmä pystyy seuraamaan helposti useita pelaajia yhtäaikaisesti. Verrattuna käsikellotettuun tilastointiin, jossa ihminen luontaisesti helpommin tekee mittausvirheitä, johtuen esimerkiksi havaitsemiskyvyn rajallisuudesta.

Joukkueen valmentajalle selkeä hyöty vaihtojen automaattisesta tunnistamisesta ja tilastoinnista on esimerkiksi pelaajakohtaisen peliajan tarkka seuranta ilman lisävaivaa tai kustannusta. Vastaavasti vedonlyöntiorganisaatioille vaihtojen ja sitä kautta pelaajakohtaisen peliajan automaattinen seuranta avaa uusia vedonlyöntimahdollisuuksia. Samanaikaisesti peliajan ja vaihtojen seuranta tuo lajin kannattajille uusia tilastoja, joista voi seurata esimerkiksi pelaajan suoritusta reaaliaikaisesti. Pelaajalle laajennettu tilastointi avaa myös uusia mahdollisuuksia seurata omaa kehittymistään.

Jääkiekon tilastointi palvelee monia sidosryhmiä, jolla on kaikilla omia vaatimuksia



Kuva 4.1 *Automaattisen tilastoinnin hyötyjä*

tilastointijärjestelmälle. Kuvassa 4.1 esitetään esimerkkejä sidosryhmien kiinnostusten kohteista automaattiseen tilastointiin. Kuvassa esitetään pelaajien kiinnostukseksi oman kehityksen seuranta ja vertailtavuus muihin pelaajiin, vastaavasti kannattajan hyötynä esitetään pelaajien ja joukkueiden vertailu. Valmentajille tilastointi tuottaa jatkuvaa kuntotietoa ja tilastoja aikaisemmin tilastoitumattomista asioista. Vedonlyönnin kannalta kuvassa 4.1 esitetään hyödyiksi automaattiset, virheettömät tilastot, jotka mahdollistavat uusia vedonlyöntikohteita.

Automaattisessa tilastoinnissa osakomponentti on vaihtojen tarkastelu ja tilastointi. Itsessään vaihtojen tilastointi on yksinkertaista, jos jääkiekon sääntöjä noudatetaan tarkasti ja vastaanotettu paikannustieto olisi virheetöntä. Ongelmia vaihtojen tunnistamiseen tuottaa sääntöjen joustava tulkinta ja virhe paikannuksessa. Vaihtoja saisi tehdä pelitilanteen aikana vain, jos molemmat pelaajat ovat 1,5 metrin etäisyydessä omasta pelaajapenkistään, eivätkä osallistu pelitilanteeseen [6] Ottelun aikana tapahtuu usein kuitenkin tilanteita, joissa pelaaja osallistuu peliin tai on huomattavasti yli 1,5 m etäisyys vaatimuksesta, kun vaihtava pelaaja palaa pelaajapenkille. Paikannustiedon kanssa ongelmaksi syntyy virhepaikannus, jota korostaa pelaajien tapa nojautua kohti pelikenttän reunusta. Paikannettava komponentti on kaudella

2017-2018 sijainnut pelaajien olkapäässä, jolloin nojailu siirtää pelaajan paikannusta kohti kaukalon reunaa ja usein jäälle.

Vaihtojen tunnistamisen ongelma on kiteytettävissä seuraavina vaatimuksina toteutettavalle komponentille:

- Latenssin pitää olla mahdollisimman alle 10 sekuntia.
- Vaihdot tunnistuvat oikein huolimatta pelaajien sääntörikkomuksista vaihtamiseen. Sääntörikkomuksia ovat esimerkiksi liian aikainen lähtövaihtoalueelta.
- Vaihdot tunnistuvat oikein huolimatta virheellisestä paikannusdatasta.

Pelaajan aktiivisuuden selvittämien on tärkeää myös muille tilastointikomponenteille, sillä esimerkiksi laukaisijan tunnistaminen vaatii luotettavaa tietoa pelissä olevista pelaajista. Muita selkeästi pelaajien oikeasta aktiivisuudesta hyötyviä komponentteja ovat esimerkiksi pelaajien keskinopeus, matka tai aika.

4.2 Peli- ja paikannustiedon kerääminen

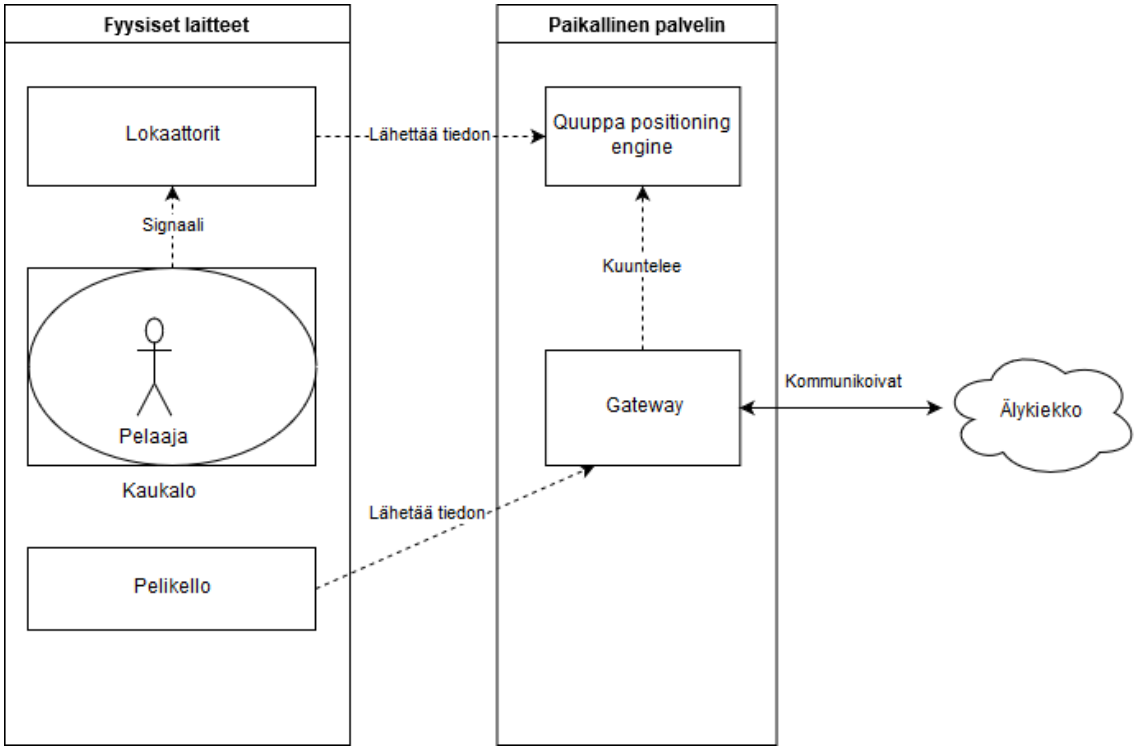
Paikannustietoa kerätään jäähalleista käyttämällä Quuppa Positioning Engineä (QPE) paikannusjärjestelmää. Yleisperiaatteena QPE-järjestelmässä on signaalia lähettävä mikrosiru, joka lähettää signaalia havainnoiville lokaattoreille. Lokaattorit tuottavata mittaustietoa paikannusta varten. Paikkaa arvioidaan kulmamittauksella vastaanotetuista signaaleista.

Paikannustiedon lisäksi otteluista kerätään tietoa pelikellosta. Kerättyyn tietoon kuuluu esimerkiksi pelikellon aktiivisuus ja aika, joita käytetään antamaan kontekstia paikannustiedolle. Kerätty tieto yhdistetään paikannustiedon kanssa yhtenäiseksi kuvaukseksi ottelun tilasta. Huomattava ongelma ottelutiedon keräämisessä on standardin puuttuminen kellosignaalille.

Kellotiedon standardoimiseksi on määritelty kellosignaali käyttämään kahta viestityyppiä, jotka ovat:

- MatchStateChanged: Pelikellon tila muuttui
- MatchClock: Pelikellon nykyinen arvo

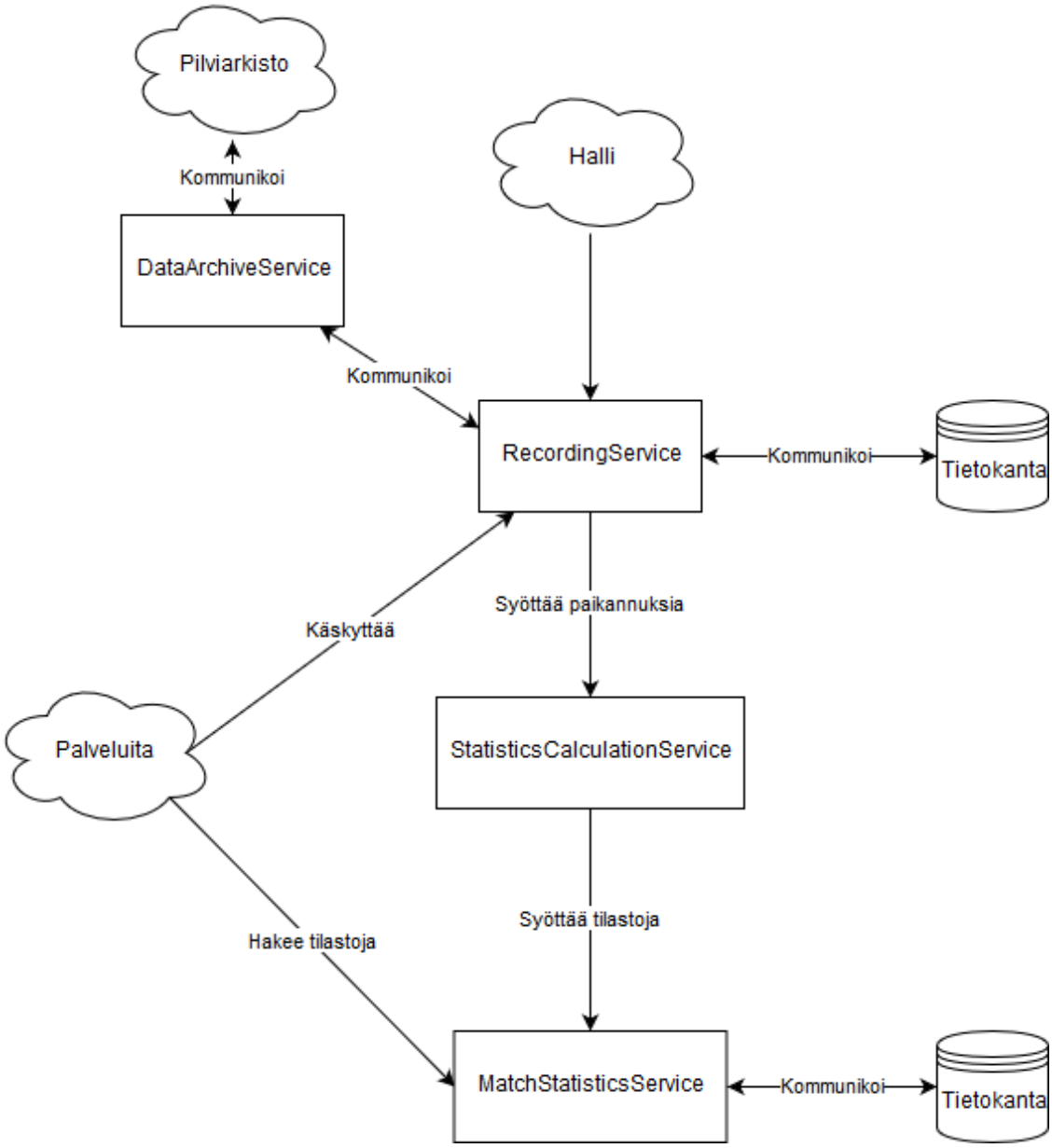
Kuva 4.2 *Paikannus ja peliteidon kerääminen*



Ottelun nykyinen tila päätellään esitettyjen kahden viestityypin perusteella. Pelikellon aktiivisuus päätellään MatchStateChanged-viestien kuvaamista tilanmuutoksista. Vastaavasti pelikellon nykyinen arvo luetaan vastaanotetuista MatchClock-viesteistä. Tällöin viestit yhdistämällä saadaan kokonaiskuva ottelun tilasta.

Kuvassa 4.2 esitellään pelitiedon keräykseen käytetyt komponentit yleisellä tasolla. Pelaajalla oleva mikrosiru lähettää signaalia lokaattoreille, jotka lähettävät tarvitut tiedot QPE:lle paikannustiedon laskemiseen. QPE tuottaa ulos paikannustietoa kolmessa ulottuvuudessa ja saatu paikannustieto sisältää ajanhetken paikannuksesta. Ajanhetki on sidottu fyysisen palvelimen kelloon.

Paikannuksia vastaanotetaan eri taajuuksilla kiekolle ja pelaajille. Jääkiekkojen paikannus vastaanotetaan huomattavasti korkeammalla taajuudella kuin pelaajien. Päätös pelaajien paikannussirujen matalammasta taajuudesta keventää paikannuksien vaatimaa suoritustehoa ja pelaajien nopeus ja paikka ei muutu tiheästi. Saatu paikannustieto vastaanotetaan ohjelmalla, joka parittaa tietoon pelikellon ajan ja ohjaa pelitilannetietoa pilvessä olevaan toteutukseen.

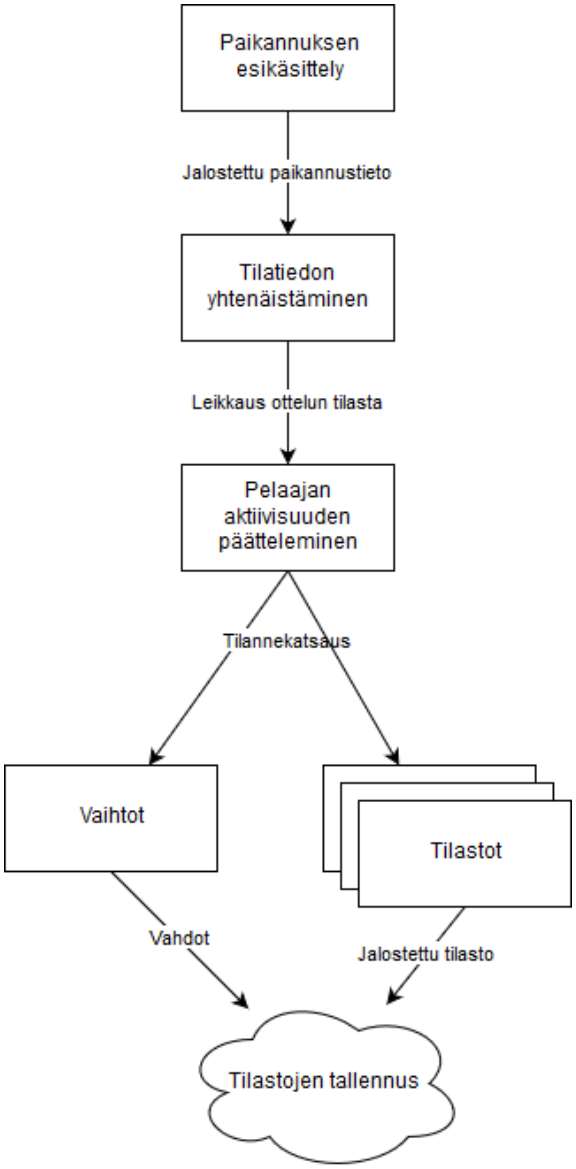


Kuva 4.3 *Tilastoinnin yleinen arkkitehtuuri*

4.3 Ottelutiedon käsittelyn arkkitehtuuri

Esitetyssä arkkitehtuurikuvassa 4.3 nauhoitusten hallintapalvelu RecordingServicen vastuualue on hallinnoida jäähallin fyysisen palvelimen kanssa kommunikaatiota. RecordingService lukee ottelutiedot hallita ja välittää hallita saamansa tiedon tilastointipalvelulle ja pidempiaikaiseen talletukseen arkistoon. RecordingService vastaa myös tilastoinnin alustamisesta.

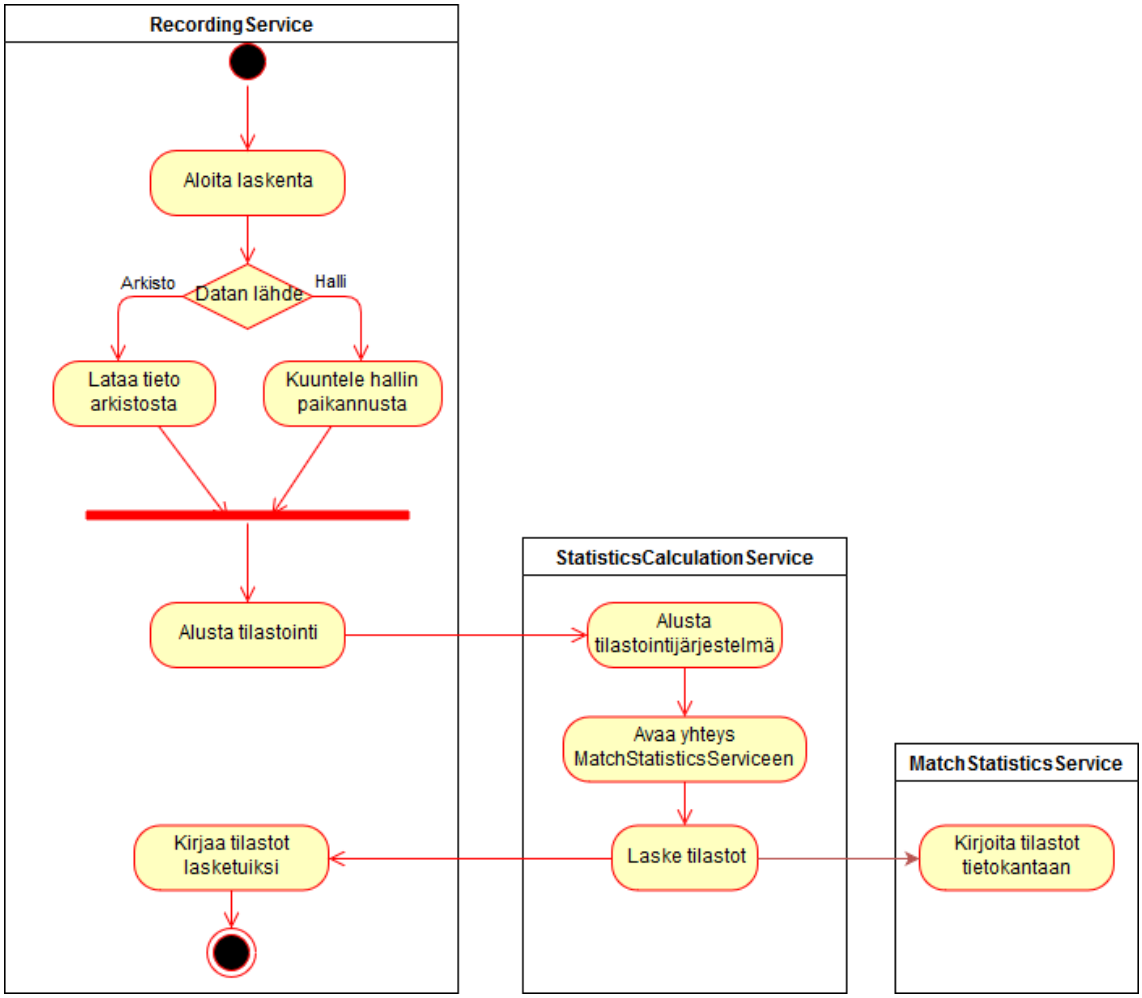
Tilastojen laskentapalvelu StatiticsCalculationSerivce on palvelu, joka laskee saamastaan paikannustiedosta vaaditut tilastot. Laskettuja tilastoja ovat esimerkiksi vaihdot, maalit ja laukaisut. Ulospäin tilastointipalvelu tarjoaa gRPC-rajapinnan,



Kuva 4.4 *Yleinen kuva tilastoinnin arkkitehtuurista*

johon voi avata gRPC datavirran. Sisäisesti palvelu on toteutettu vuoarkkitehtuurilla käyttäen Dataflow kirjastoa.

Kaaviossa 4.4 esitetään karkea kuvaus tilastoinnin arkkitehtuurista. Tilastoinnin komponenteissa esitetään paikannuksen esikäsittely, jolla pyritään poistamaan kohinaa ja virhepaikannuksia saadusta paikannustiedosta. Tilatiedon yhtenäistämisessä kerätään tilannekatsaus ottelun yksittäisestä hetkestä. Tilannekatsaukseen kerätään esimerkiksi 0,02 sekunnin ajalta kaikki saadut paikannukset, kellolaitteiston viimeisin tilatieto ja muuta olennaista kerättyä ottelutietoa. Tilannekatsauksen luonnissa oleellinen komponentti on pelaajan aktiivisuuden selvittäminen. Kuvassa 4.4 esitetään kuinka tilannekatsauksen luonnin jälkeen tilastoinnin vuo haaraantuu useaan haaraan, joita ovat esimerkiksi laukausten tunnistaminen tai vaihtojen tunnistami-



Kuva 4.5 *Vuokaavio tilastojen laskennan aloittmisesta*

nen. Viimeisenä osana tilastointia on lähettää lasketut tilastot eteenpäin. Kuvan 4.3 mukaisesti tilastointipalvelu ottaa yhteyden MatchStatitcsServiceen, joka kirjoittaa tiedot tietokantaan ja tarjoaa rajapinnan tietojen hakua varten.

Kuvassa 4.5 esitetään tilastoinnin toiminnan vuo. Recording service vastaanotettuaan käskyn aloittaa laskennan, sekä selvittää käytetäänkö laskentaan arkistoon tallennettua tietoa ottelusta vai jäähallilta saatavaa reaaliaikaista tietoa. Selvitettyään tiedon lähteen RecordingService kutsuu StatisticsCalculationServicen rajapintaa alustaen vuon, jonka läpi siirtyy tietoa palvelulle. Sisäisesti StatisticsCalculationSerivce rakentaa tilastointiin tarpeeliset komponentit ja avaa gRPC yhteyden MatchStatisticsServiceen. RecordingServicen avaamasta vuosta tilastointi-palvelu laskee tilastoja, joita palvelu lähettää edeleen MatchStatiticsService:lle. Tilastojen laskennan loputtua RecordingService merkitsee ottelun tilastoinnin loppuneeksi.

4.4 Paikannusvirheen kompensointi

Vastaanotetussa paikannustiedossa on huomattavaa kohinaa. Paikannusvirhettä syntyy luontaisena virheenä kulmamittauksen tarkkuudesta, hetkellisistä katvealueista tai mahdollisesta signaalin muusta häiriytymisestä. Kaikki mainitut tekijät aiheuttavat vastaanotettuun paikannukseen kohinaa, joka vaikeuttaa pelaaja tarkan sijainnin määrittämistä.

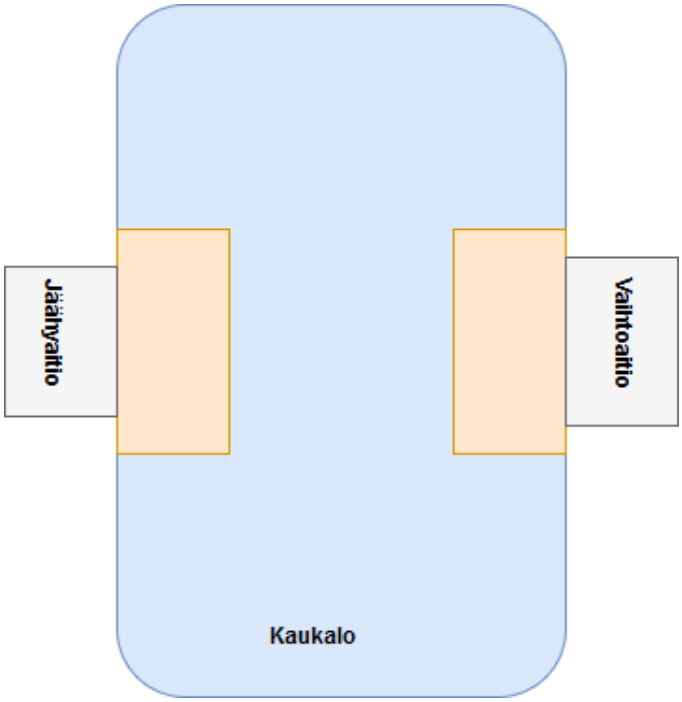
Vaihtojen tunnistamisen kannalta olennaista on pystyä määrittämään tarkasti, milloin pelaaja on vaihto- tai jäähyaitioissa verrattuna milloin pelaaja on kentällä. Ongelmaksi muodostuu esimerkiksi pelaajien inhimillinen tapa nojautua kohti pelikenttää. Ongelma muodostuu, kun pelaajien nojailuun lisätään kohina, jolloin usein pelaaja näkyy pelikentällä huolimatta oikeasta sijainnista vaihtoaitioissa. Ongelma peilautuu vastaavasti, jos pelaaja esimerkiksi taklataan pelaaja- tai vaihtoaitioon, jolloin pelaaja paikantuu esimerkiksi vaihtoaitioon, vaikka pelaajan kuuluisi pysyä pelikentällä.

Pelaajan oikean tilan ennustamiseen tarvitaan siis tietoa pelaajan paikannuksen historiasta, eli kuinka pelaaja on päätynyt nykyiseen pisteeseensä. Ongelma pelkästään historiaan katsomisessa on, että paikannustiedosta on haastavaa päätellä esimerkiksi, onko pelaaja luistellut vaihtoaitioon vai luistelee vaihtoaition vierestä. Tällöin pelaajan paikan selvittämiseen pitää tutkia myös pelaajan tulevia liikkeitä. Pelaajan tulevan paikan selvittämiseen joudutaan käyttämään puskurointia, joka johtaa latenssiin järjestelmässä.

Tarkastelemalla pelaajan tulevaisuuden paikkaa, saadaan huomattavasti tarkempi arvio pelaajan liikkeestä, sillä voidaan tarkastella pelaajan lyhyen ajan käyttäytymistä. Käyttäytymiseen voidaan tehdä oletuksia kuten pelaajan paikannukset peliaitoon ovat todennäköisemmin totta, jos pelaaja siirtyy pelialueella pidemmälle. Tällöin pelaajan tilaa mallintaa luontaisesti hystereesillä, jossa pelaajan tila määräytyy seuraavan tulevaisuudessa olevan selkeän tilan perusteella.

Kuvassa 4.6 esitetään kaukalon aluejako vaihtojentunnistamiseen liittyvin osin. Kaukalossa sekä jäähy-, että vaihtoaition edessä on alue, jonka sisällä olevista pisteistä on haastavaa tehdä pelkän paikannuksen perusteella päätelmää pelaajan oikeasta tilasta. Mikäli paikannus olisi tarkkaa, molemmat raja-alueet olisivat selkeästi jäällä. Paikannuksen virheellisyyden takia joudutaan sijaintia arvioimaan molemmissa varoalueissa. Sekä jäähyaitoon että vaihtoaitioon pätevät samat ongelmat.

Paikannuksen virhetilan syntymiselle on useita syitä. Alla olevassa listauksessa esitetään kolme yleisintä syytä paikannuksen virhetilan synnylle.

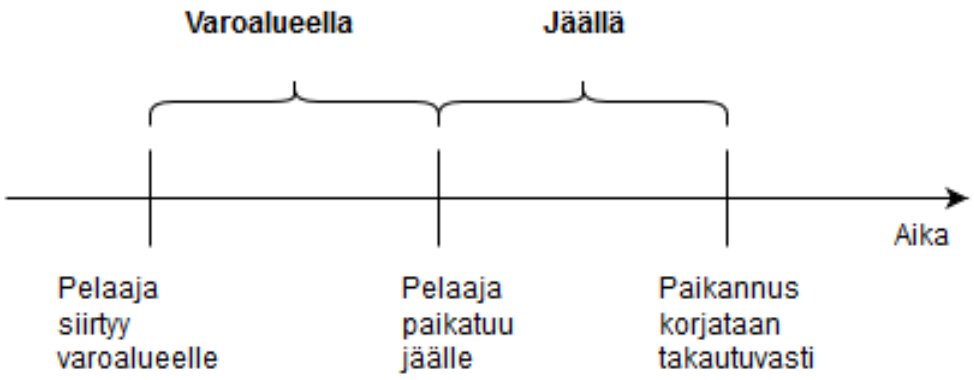


Kuva 4.6 *Kaukalon alueet*

- Pelaaja voi virheellisen paikannuksen vuoksi paikantua väärälle alueelle.
- Pelaaja voi paikantua kohinan vuoksi väärälle alueelle.
- Pelaaja voi olla fyysistesti väärällä alueella.

Kontekstissa kohina määritellään pienenä poikkeamana oikeasta paikannuksesta. Puolestaan virheellinen paikannus on selkeä poikkeama yleisessä sijainnissa. Viimeinen tapaus kuvaa tilanneetta, jossa esimerkiksi pelaaja on taklattu fyysisesti jäähy- tai vaihtoaitioon. Ratkaisuna selkeisiin virhepaikannuksiin on niiden hylkääminen täysin järjestelmästä. Kohinasta johtuva virhe samanaikasesti pelaajaan fyysiseen virheelliseen sijainnin kanssa joudutaan ratkaisemaan yllä esitetyllä mallilla eli tulevien pisteiden sijaintitiedolla.

Huomattavaa on, että vaihtoaition edessä olevat ongelmat ovat haastavampia kuin jäähyaition varoalueen ongelmat. Johtuen jääkiekon säännöistä on selkeää, että kentältä ei saa pelaaja siirtyä jäähyaitioon, kun ottelu on käynnissä. Vastaavasti jäähyaitiosta ei kentälle saa siirtyä pelaajia ennen rangaistuksen loppumista. Molemmat ehdot helpottavat huomattavasti pelaajien tilojen tarkkailua, sillä logiikkaa voidaan yksinkertaistaa tarkkailemaan pelikellon tilaa, jolloin ei ole tarvetta tarkastella tulevia tiloja.



Kuva 4.7 *Aikajan takautuvasta tilankorjauskesta*

Yleinen ratkaisu jäällä olemisen selvittämiseen vaihtoaition edessä perustuu hypoteesiin: ”Jos pelaaja paikantuu varoalueelle, pelaajan tila on todennäköisesti ollut se tila, johon pelaaja paikantuu, kun varoalueelta poistutaan”. Tapauksessa, jossa pelaaja on lyhyen ajanhetken varoalueella, hypoteesi on tarpeeksi tarkka approksimaatio oikeasta tilanteesta. Ongelmatilanteeksi nousevatkin tilanteet, joissa pelaaja paikantuu pitkiä aikoja varoalueelle. Esimerkisi ongelmatilanne voi nousta, jos pelaaja paikantuu varoalueelle kohinan takia. Tällöin pelaaja lähtiessään varoalueelta jäälle, paikantuu takautuvasti väärin. Vastaava ongelmatilanne on, jos pelaaja esimerkiksi taistelee varoalueella kiekosta ja taklautuu vaihtoaitioon, jolloin pelaajan tila korjattaisiin takautuvasti väärin.

Ratkaisu yksittäisten pisteiden aiheuttamaan ongelmaan on selkeä, vaaditaan jonkin yhtäjaksoinen määrä pisteitä joko jäällä tai vaihtoaitiossa. Asettaessa kynnysarvo tulonsiirrolle saadaan estettyä nopeat heilahtelut, mutta vastaavasti maksetaan hinta pidemmästä latenssista yksittäisen pisteen tilanselvityksessä. Kuvassa 4.7 esitellään aikajana, jossa pelaaja on siirtynyt varoalueelle ja siellä jäälle.

Kompensoimalla yksittäisten pisteiden tuottamia virhehavaintoja tarkastelemalla tulevaa tilaa, syntyy toinen ongelma johtuen systemaattisesta virheestä ja kohinasta. On yleistä, että pelaaja paikantuu pitkäksi yhtäaikaiseksi jaksoksi kaukalon sisälle, vaikka pelaaja oikeasti sijaitsee joko vaihto- tai jäähyaitiossa. Tilannetta monimutkaistaa kohina, joka saa pelaajan paikannuksen vaikuttamaan liikkeeltä. Ratkaisuksi esitetään hypoteesia: ”Jos pelaaja on varoalueella paikallaan, pelaaja ei ole jäällä”. Kappaleen paikallisuuteen voidaan esittää esimerkiksi paikannuksen varianssin tarkastelua tai pelaajan nopeutta.

Kuvassa 4.8 esitetään pelaajan tilan arviointi käyttämällä hystereesiä. Hystereesissä seurataan pelaajan nykyistä tilaa ja vaaditaan ennen tilasiirtymää, määritetty määrä yhtäjaksoista paikannusta vastakkaiseen tilaan. Mikäli pelaaja paikantuu nykyiseen

Kuva 4.8 *Hystereesi pelaajan tilan mallintamiseen*

tilaan, tai varoalueella nollataan laskuri. Hystereesin herkkyys määrittää kuinka pitkän ajan pelaajaan tarvitsee paikantua vastakkaiseen tilaan, jotta pelaajan tila vaihtuu.

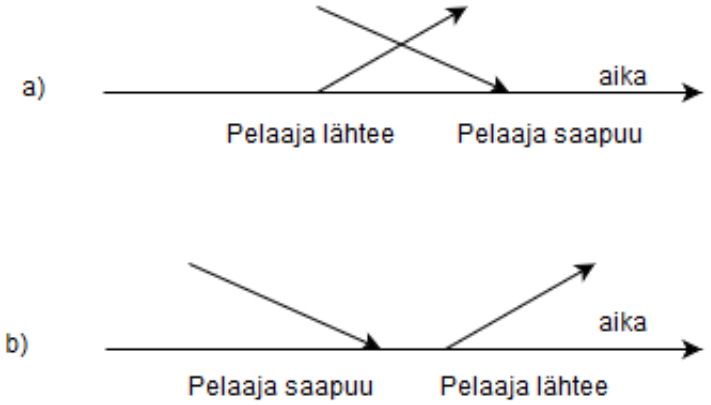
Paikannustietoa vastaanotetaan virtana leikkauksia kaukalon tiloista. Kohinan kompensoinnin tavoitteena on leimata tilaleikkauksiin pelaajien kohinaton tila ja lähettää leikkaukset eteenpäin virtana. Ongelmallista virtana käsiteltävässä datassa on, että koko ottelun tila ei ole saatavissa välittömästi, vaan joudutaan puskuroimaan tietoa kompensoinnin mahdollistamiseksi. Toisena ongelmana on, että historiaa ei voi puskuroida kuin äärellisen ajan taatakseen muun järjestelmän toimivuuden, sillä kompensoitu tilatieto tarvitsee lähettää muille komponenteilla datavirtana, jota ei järjestelmän rinnakkaisuuden takia saa muuttaa jälkeenpäin.

Pelaajan tilan jälkikäteistä korjausta varten on toteutuksessa tarpeellista tehdä tilatiedon puskurointia. Puskurointi mahdollistaa retroaktiivisen tilakorjauksen, mutta kasvattaa ohjelmistokomponentin vasteaikaa. Puskuroinnin pituuden sääteleminen vaikuttaa maksimiaikaan, jonka komponentti voi puskuroida ja samalla vaikuttaa kuinka pitkän ajan perustella päätös tilasta voidaan tehdä tai korjata.

4.5 Vaihtojen parittaminen

Jääkiekkokon sääntöjen mukaan kentällä ei saa olla aktiivisena normaalissa tilanteessa kuin viisi pelaajaa. Kuitenkin pelialueella on usein useampi kuin viisi pelaajaa. Tilastoinnin kannalta on haluttua, että pystytään selvittämään tilanne yksiselitteisesti sääntöjä noudattavaksi tilaksi. Tilannetta muodostaa ongelmalliseksi esimerkiksi vaihteleva määrä pelaajia kentällä johtuen sääntöjen mukaisista rangaistuksista tai muista pelitilanteista [6].

Ongelma vaihtojen parittamisessa on ihmislähtöinen. Sääntöjen mukaisessa tapauksessa pelaajan kuuluu pysyä 1,5 metrin säteellä vaihtoaitiosta ennen korvaavan pelaajan saapumista. Käytännön tilanteissa pelaajien saapumiset ja poistumiset vaih-



Kuva 4.9 *Yleiset tapaukset pelaajien vaihdoissa*

toaitioon eivät noudata sääntöjen vaatimaa ideaalista tapausta, vaan pelaajat voivat siirtyä kentälle pelaamaan samalla kuin vaihtoaitioon saapuva pelaaja voi olla vasta luistelemassa huomattavasti vaaditun rajan ulkopuolella.

Kuvassa 4.9 esitetään yleiset vaihtoedot vaihtojen tapahtumiselle. Tapauksessa *a)* pelaaja lähtee vaihtoaitiosta ennen korvaavan pelaajan saapumista. Käänteisesti tapauksessa *b)* pelaaja saapuu vaihtoaitioon ennen vaihtavan pelaajan lähtöä. Tavoitteena on parittaa tapahtumat toisiinsa, siten että kentällä on aktiivisena oikea määrä pelaajia. Tavoitteellisesti tapauksessa *a)* aktiivisena on yhtäaikaisessa ajassa jäälle lähtenyt pelaaja. Vastaavasti tapauksessa *b)* aktiivinen pelaaja on vaihtoaitioon saapuva pelaaja.

Yleisessä tapauksessa aika pelaajan saapumisen ja lähtemisen välissä on lyhyt, mutta ongelmatapaukseksi muodostuvat tilanteet, joissa pelaajien lähtemisen ja saapumisen väli on pitkä. Ongelmallisia tilanteita ovat esimerkiksi, jos pelaajalla ei ole paikanninta lainkaan tai paikannin on vikatilassa. Vikatilojen ulkopuolella ongelmallisia tilanteita ovat esimerkiksi, jos korvaava pelaaja ei saavu vaihtoaitioon. Esimerkiksi pelaaja voi saapua vaihtoaitioon vasta pelikatkolla, jolloin joukkue saattaa tehdä useampia vaihtoja, tehden vaihtaneen pelaajan seuraamisesta haastavaa.

Haaste pelaajien vaihtojen parittamisessa on päätellä pelaajat, joiden vaihtoja seurataan. Käytännössä valinta seurattaviin pelaajiin tehdään aloituksien yhteydessä, sillä pelaajien oikean määrän arviointi on huomattavasti vaikeampaa automatisoida, sillä määrä voi vaihdella maalivahti mukaanlukein neljän ja kuuden välillä välillä riippuen erästä tai kaudesta. Sillä aloituksessa pelaajien tilat päivitetään aina, ei toteutuksessa seurata pelaajien tiloja lainkaan pelin ollessa tauolla.

4.6 Testaaminen

Testaamisessa keskitytään testaamaan vaatimuksissa tärkeimpiä vaatimuksia eli selvittämään järjestelmän latenssin käyttäytyminen suhteessa tilastojen oikeellisuuteen. Testien läpäisyyn vaikuttavia komponentteja ovat väärät tulkinnat vaihdoista ja tulkintaan käytetty aika. Samanaikaisesti toteutetaan keskiviiveen laskenta koko tilastointi järjestelmälle saadakseen kokonaiskuvaa yksittäisen komponentin aiheuttamasta latenssista suhteessa muihin komponentteihin. Sillä testit keskittyvät vaatimusten tarkkailuun ja järjestelmän kokonaisvaltaiseen toimintaan, testaaminen voidaan luokitella järjestelmätestaamiseksi. Samanaikaisesti testit testaavat yksittäisen komponentin toimintaa, jolloin testejä voidaan myös pitää integraatiotesteinä.

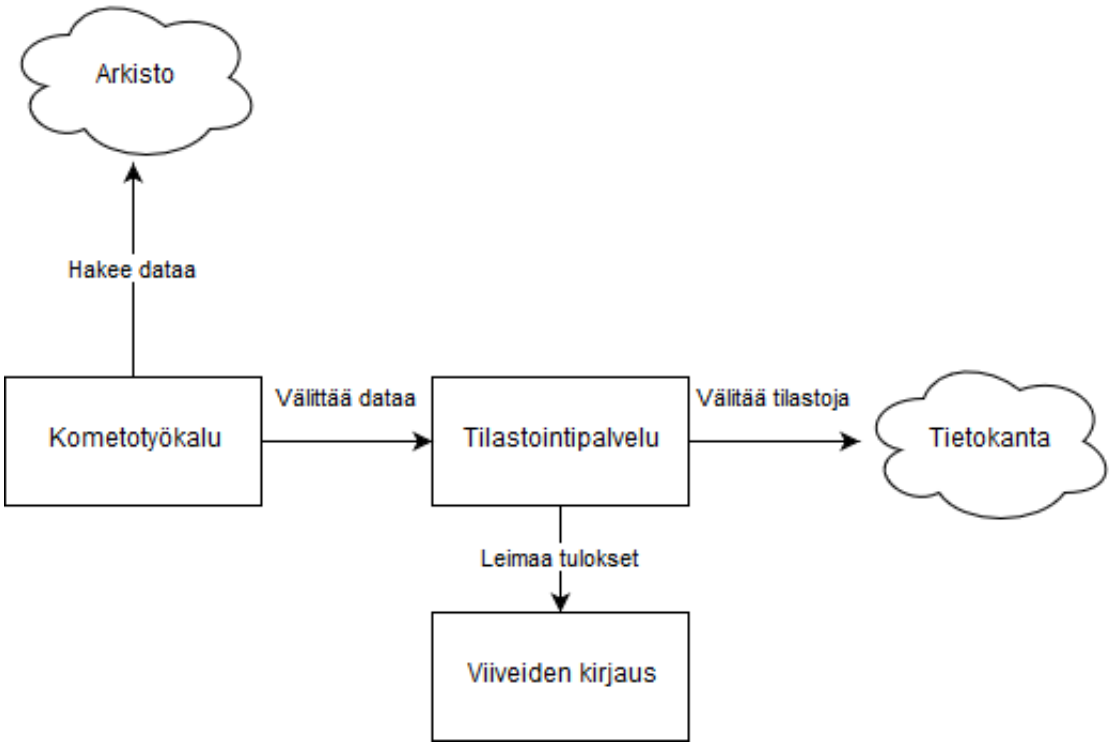
Yleistä arkkitehtuuria esittävässä kuvassa 4.5 esitetään yleinen arkkitehtuuri ja ottelutallenteen käsittelyn vuo. Testaustilanteessa keskitytään käsittelemään arkistoon tallennettu ottelu laskemalla ottelun tilastot uudestaan. Testausajosta tallennetaan konfiguraatio, jolla testi on ajettu ja verrataan saatuja tuloksia muilla konfiguraatiolla tehtyihin ajoihin.

Keskimääräinen latenssi saadaan laskettua tilastokohtaisesti tallentamalla jokaiselle datapisteelle aika, jolloin se syötettiin tilastoinnille, vastaavasti merkitään jokaiselle tilastolle laskemisajankohta. Vertailemalla tilastojen laskemisajankohtia suhteessa vastaavan pelihetken paikkatietoon saadaan laskettua latenssi. Latenssista voidaan myös huomioida latenssin käyttäytyminen peliajan suhteen.

Vaihtojen laadulliseen tarkkailuun käytetään niiden vaihtojen määrää, jotka eivät lopu pelikatoon ja ovat alle 5s pitkiä. Syy valittuun mittariin on sen helppous mittauksessa, lisäksi on todennäköistä, että vaihto on havaittu johtuen kohinasta ja osoittaa virheen läsnäolon järjestelmässä. Lisäksi numeerinen mittari sallii vertailun eri parametrin vaikutuksesta tuloksiin.

Testaamisessa simuloidaan ottelun tilastojen laskentaa käyttämällä pelatun ottelun paikannus- ja kellotilatietoa. Viiveen laskennan toiminnan takaamiseksi datavuota keinotekoisesti hidastetaan kuvaamaan tuotantotilanteen vuon taajuutta. Jos vuota ei keinotekoisesti hidastettaisi, niin latenssit eivät vastaisi todellisuutta. Hidastaminen tapahtuu vertailemalla syötteiden aikaleimoja ja tarvittaessa odottamalla ennen seuraavan syötteen syöttämistä.

Kuvassa 4.10 esitetään arkkitehtuuri testiottelun ajamisesta. Kuvaassa komentotyökalu hakee datavuon arkistosta ja uudelleen ohjaa saamansa syötteet kohti tilastointipalvelua. Tilastointipalvelu välittää vastaanottamansa tiedot eteenpäin talletettavaksi tietokantaan. Erona normaaliin ottelun ajoon on syötteiden leimaaminen.



Kuva 4.10 *Testiajojen kuvaus*

Datavirran aikaleima	Pakannuksen syöttöhetki	Vaihdon ulostulo	Latenssi
0	50	70	70 - 50 = 20
20	70	-	-
80	100	130	30

Taulukko 4.1 *Esimerkki viiveiden leimaamisesta*

Testiajossa datavirrasta saadut pisteet leimataaan suorittavan tietokoneen sisäisellä kellolla. Leimatut pisteet tallennetaan siten, että tiedetään jokaista datavirran aikahetkeä vastaava aika laskentaa suorittavan koneen ajassa. Vastaavasti lasketut tilastot leimataan sisäisellä kellolla, jotta latensseja voidaan vertailla yksinkertaisesti. Sisääntulot ja ulostuloja verrataan datavirran aikaleimojen mukaisesti, jolloin viiveet ovat laskettavissa.

Taulukossa 4.1 esitetään esimerkki kuinka dataa käsitellään viiveen laskennan yhteydessä. Samalle datavirran aikaleimalle haetaan sekä paikannuksen sisäänsyötön aikaleima, että tilaston ulostulon aikaleima. Viive lasketaan yksinkertaisesti aikaleimojen erotuksena. Huomattavaa on, että tilastoja ei ole testeissä laskettavissa jokaisella aikaleimalle.

Testauksessa on oletettua virhettä, joka johtuu esimerkiksi syötteiden ja ulostulon leimaamisen tarpeesta, joka tuo järjestelmään lisäkuormaa. Vastaavasti toinen viiveenlaskentaan liittyvä ongelma johtuu suoritusjärjestyksen epävarmuudesta, jolloin

tilastoja ei leimata välittömästi, kun ne on laskettu. Samalla laskettu viive on suurin mahdollinen, eli vertailtavat pisteet ovat aikaisin aikaleimalle saatu paikannus, suhteessa viimeiseen aikaleimaan liittyvään tilastoon.

5. TULOKSET

Tässä luvussa esitetään mittaussuunnitelma tulosten mittaamiselle osiossa 5.1. Osiossa 5.2 esitetään mittaustulokset toteutetulta ohjelmistokomponentilta. Osiossa 5.3 esitetään johtopäätöksiä perustuen esitettyihin mittaustuloksiin. Lopuksi luvussa ehdotetaan jatkokehityskohteita ja -toimenpiteitä.

5.1 Mittaussunnitelma

Työssä on toteutettu ohjelmistokomponentti, joka laskee pelaajien tekemät vaihdot, sekä määrittämällä pelaajan aktiivisuuden paikannuksen perusteella ja seuraamalla pelaajien tilojen muutoksia. Komponentti on integroitu osaksi laajempaa tilastointijärjestelmää, jossa sen toimintaa mitataan. Testaus noudattaa esitettyä testaussuunnitelmaa, jossa mittaan latenssia ja todennäköisesti virheellisten vaihtojen määrää.

Mittauksissa säädellään vaihtologiikaan liittyviä parametrejä, jotka ovat sisäisen puskuroinnin pituus ja sisäisen tilaa mallintavan hystereesin muutosherkkyys. Muutosherkkyys määrittää kuinka herkästi havaittu pelaajan tila siirtyy aktiivisen ja epäaktiivisen välillä. Sisäisen puskuroinnin pituus rajoittaa kuinka pitkän ajan toteutettu komponentti voi puskuroida dataa tehdäkseen päätöksen.

Latenssia mitataan kahdelta ottelulta, toiselle testiottelulle muutetaan edellä mainittujen parametrien arvoja. Parametrin vaikutusta latenssiin ja ohjelmistokomponentin tulosten tarkkuuteen testataan kolmella eri arvolla: oletusarvoilla, suuremmalla ja pienemmällä parametrin arvolla. Parametrien vaikutusta ei mitata yhtäaikaisilla muutoksilla oletusarvoista, eli mittauksissa hystereesin herkkyyttä ja puskurin pituutta ei ole muutettu samanaikaisesti. Mittauksia parametrien hienosäätöön viiveen laskennan kanssa ei työssä toteuteta, sillä yksittäisen testiajon suorittaminen viiveen laskennan vaatimalla keinotekoisella hidastuksella on työn tehtävän kannalta tarpeettoman hidasta. Työssä rajataan mittaukset esitettyyn parametrin kolmeen mittauspisteeseen, sillä mittauksilla on tarkoitus saada riittävä käsitys parametrin muutoksen vaikutuksesta viiveeseen ja tuloksiin.

Parametri	Keskiarvo (s)	Varianssi (s)	Minimi (s)	Maksimi (s)
Pohjatila	6.12	0.04	5.09	8.09
Pitkä puskurointi	9.67	0.24	9.07	11,9
Lyhyt puskurointi	5.63	0.28	5.03	7.09
Epäherkkä hystereesi	6.12	0.05	5.09	7.09
Herkkä hystereesi	6.15	0.09	5.11	9.16

Taulukko 5.1 *Vaihdoille mitattuja latensseja*

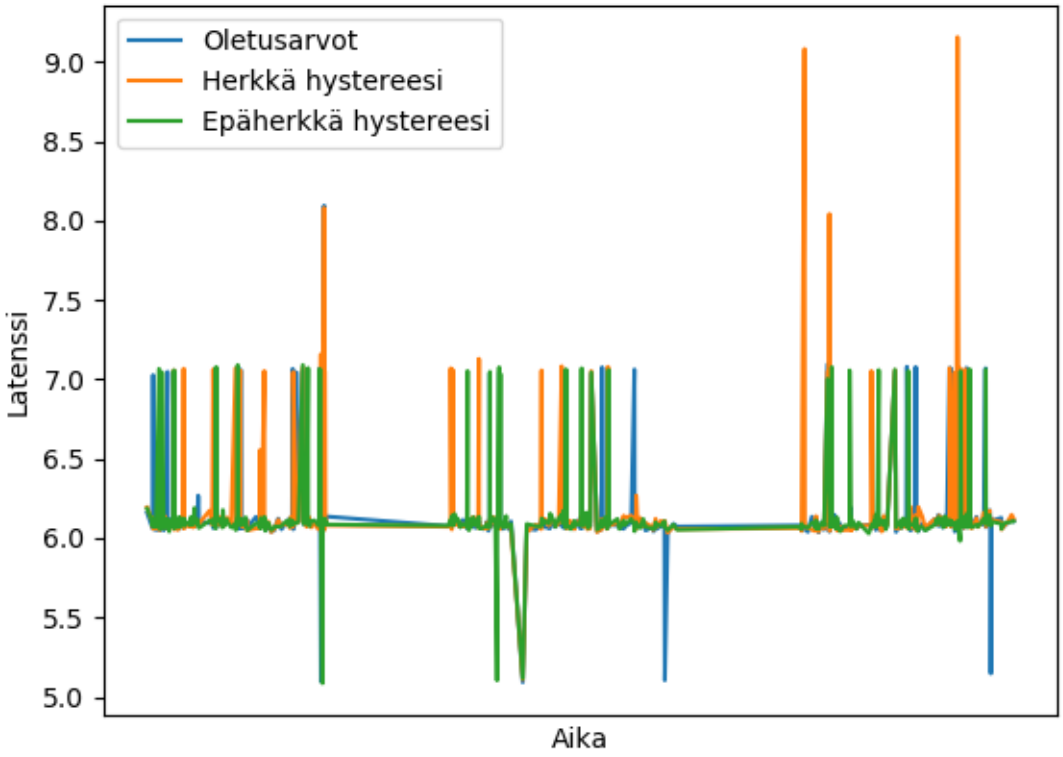
Oletuksena parametrien säätämien vaikuttaa latenssiin ja virheellisiin havaintoihin. Lyhentämällä puskuria saavutetaan pidempi vasteaika, mutta tarkkuus pelaajan tilan määrittämisessä heikkenee. Vastaavasti puskurin pidentäminen pidentää viivettä ja parantaa tarkkuutta vaihtojen tunnistamisessa. Samankaltaisesti sisäisen hystereesin herkkyyden kasvattaminen oletetaan lyhentävän viivettä, mutta riskinä on aiheuttaa huomattavasti korkeampia määriä vääriä tilasiirtymiä.

5.2 Mittaustulokset

Työssä on toteutettu ohjelmistokomponetti, joka laskee pelaajien vaihtoja paikannusdatasta. Vastaavasti työssä on esitetty suunnitelma kuinka toteutetun komponentin toimintaa testata ja mitataan. Osiossa esitettään työssä toteutetun komponentin tuloksia.

Taulukossa 5.1 esitettän mittaustuloksia toteutetulle ohjelmistokomponenteille. Taulukossa esitetään säädetty parametri ja mittaustuloksia vaihtojen laskennalle. Mittaustuloksista on valittu esitettäväksi latenssin keskiarvo, varianssi, minimi ja maksimi. Mittareiden perusteella voi yleisellä tasolla arvioida toteutetun komponentin parametrin muutoksen vaikutusta tuloksiin. Esitetyt tulokset on pyöristetty kahden desimaalin tarkkuuteen. Taulukon 5.1 tuloksissa on huomioitavaa, että latenssia järjestelmään syntyy muista komponenteista kuin vain työn mittakaavassa toteutetuista komponenteista. Mittauksissa on työn kannalta alaraja latenssille, vastaavaa ylärajaa ei ole.

Taulukon 5.1 tuloksista on havaittavissa, että hystereesin herkkyyden lisääminen kasvattaa latenssia. Testiottelulle epäherkkä hystereesi tuottaa hystereesin herkkyyden säädöistä lyhyimmän maksimi ja minimi latenssin. Varianssi on hieman suurempi kuin pohjatilan vastaava, mutta pyöristystarkkuudesta johtuen keskiarvo latenssille on sama kuin pohjatilalla. Oletuksien vastaisesti hystereesin asettaminen herkäksi tuottaa pidemmän latenssin kuin oletustilassa. Samalla herkäksi säädetyllä hystereesillä on huomattavasti suurempi keskihajonta ja hieman suurempi keskiarvo.

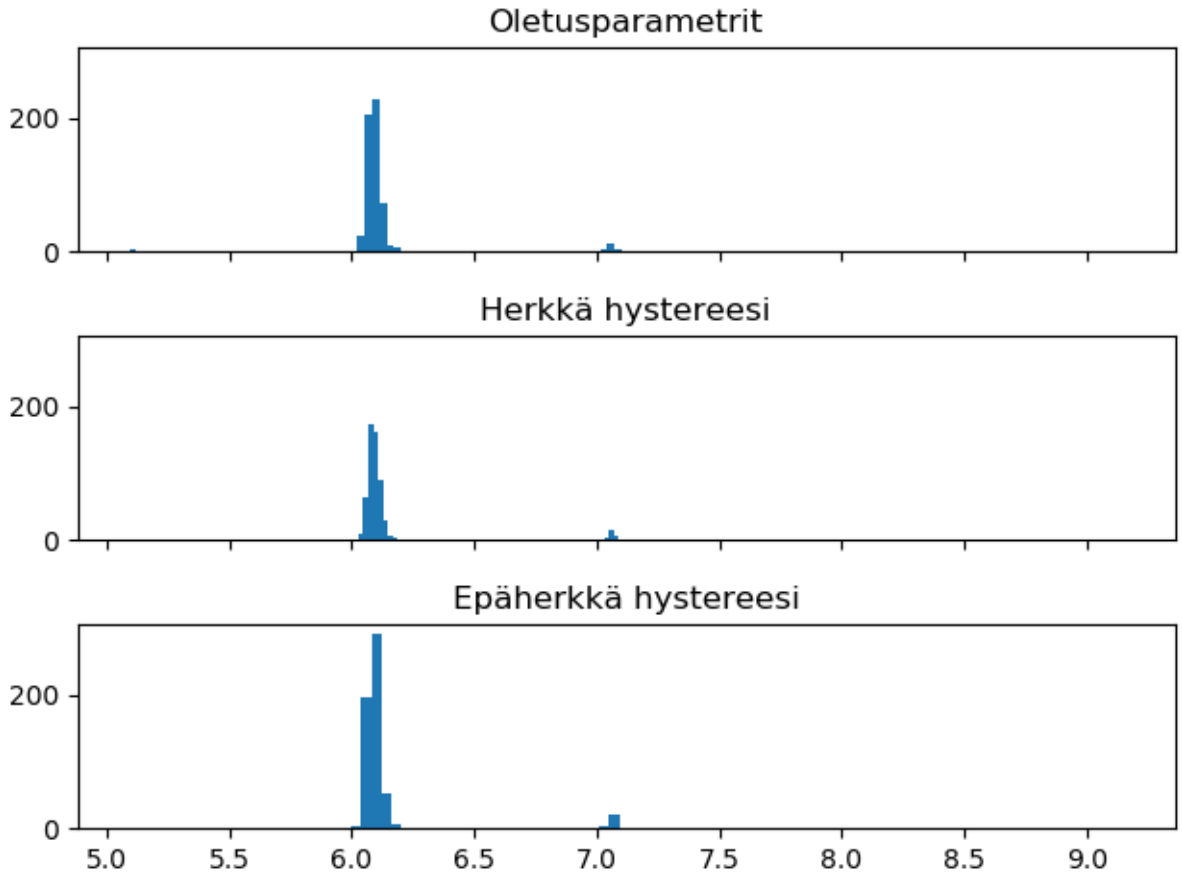


Kuva 5.1 *Hystereisn herkyyden muuttamisen vaikutus latenssiin ajassa*

Taulukossa 5.1 esitetään myös puskuroinnin pituuden säätämisen vaikutus latenssiin. Puskuroinnin kasvattamisen vaikutus toimii mittaussuunnitelmassa tehtyjen oletusten mukaisesti, jolloin pidemmällä puskurilla latenssi on huomattavasti pidempi. Puskurin pidentäminen kasvattaa myös keskihajontaa. Vastaavasti puskurin lyhentäminen kasvattaa keskihajontaa. Oletusten mukaisesti lyhyt puskurointi lyhentää latenssin keskiarvoa.

Kuvaajassa 5.1 esitetään hystereesin herkkyyden muuttamisen vaikutus latenssiin. Kuvaajassa on selkeästi havaittavissa pelikatkot ja ottelun kolme erää, joiden välillä järjestelmän latenssilla ei ole merkitytystä. Kuvaajaan 5.1 on piirretty yhtäaikaisesti sekä pohjatila, herkäksi säädetyn hystereesin tulokset ja epäherkän hystereesin tulokset. Kuvaajassa on näkyvissä järjestelmän latenssin käyttäytyminen, jossa latenssi pääasiallisesti on lyhyempi ja epäsäännöllisesti syntyy yksittäinen piste, jolle latenssi on korkeampi.

Vierailullisesti kuvaaja 5.1 osoittaa, että samalla puskurin pituudella jokainen hystereesin arvo värähtelee lähes saman tason ylä- ja alapuolella. Oletustilan paramet-



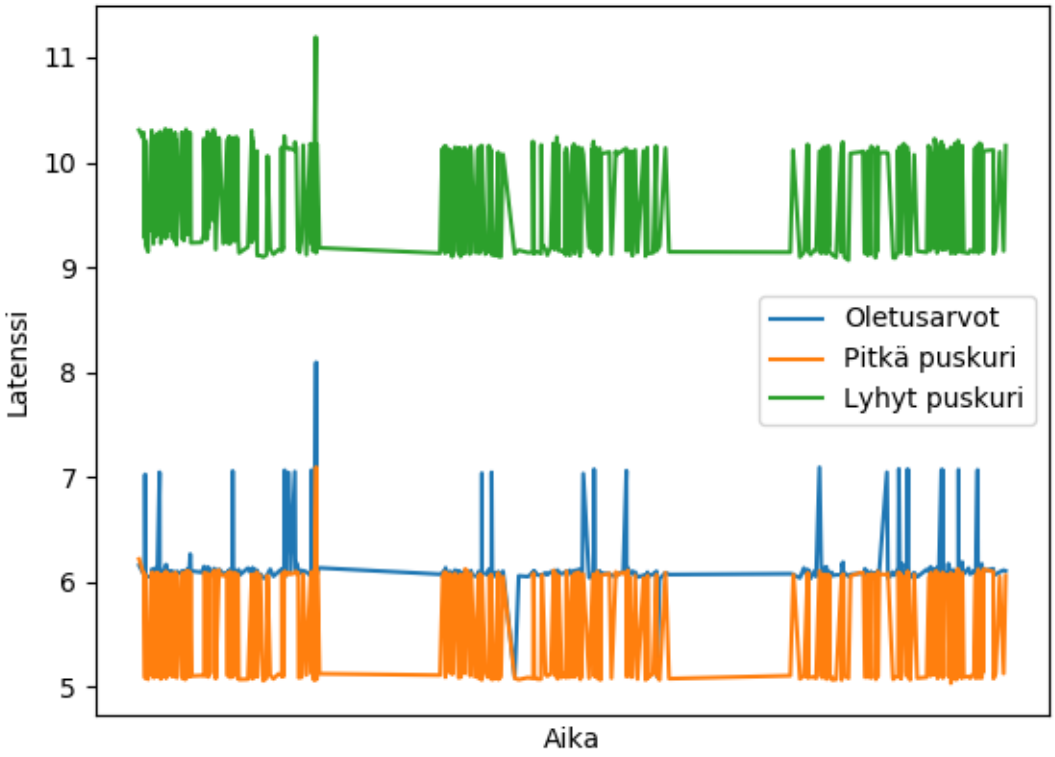
Kuva 5.2 *Histogammi lantessin käyttäytymisestä hystereesiä säädettäessä*

reillä latenssi käyttäytyy kuvaajan perusteella vähemmän poikkeavasti. Huomattavaa herkäksi säädetyn hystereesin tuloksissa on tiheämmin poikkeavia pisteitä. Samalla herkäksi säädetyn hystereesin pisteet poikkeavat suurempia määriä, tosin on huomattavaa, että suuret poikkeamat ovat yksittäisiä pisteitä.

Kuvaajasta 5.1 voi myös havaita latenssin satunnaisesti poikkeavan yksittäisissä pisteissä normaalitasosta huomattavasti alas. Pisteisiin liittyy kuvaajan perusteella alkava pelikatko. Kuvaajan 5.1 perusteella poikkeamia esiintyy vain oletusparametreillä tai jos hystereesiä on säädetty epäherkemmäksi. Herkäksi säädetty hystereesi ei poikkea latenssista alaspäin vastaavasti.

Kuvaajassa 5.2 esitetään histogrammi latenssin käyttäytymisestä, kun hystereesin herkkyyttä säädetään. Kuvaajassa esitetään mittaukset oletusarvoislta parametreiltä, herkäksi säädetyltä hystereesiltä ja epäherkäksi säädetyltä hystereesiltä. Kuvaaja 5.2 on histogrammi kuvaajan 5.1 mittauksista.

Kuvaajasta 5.2 voi havaita, että hystereesin herkkyys vaikuttaa latenssin leviämiseen oletusarvosta. Epäherkällä hystereesillä arvot ovat pakkautuneet pienemmälle

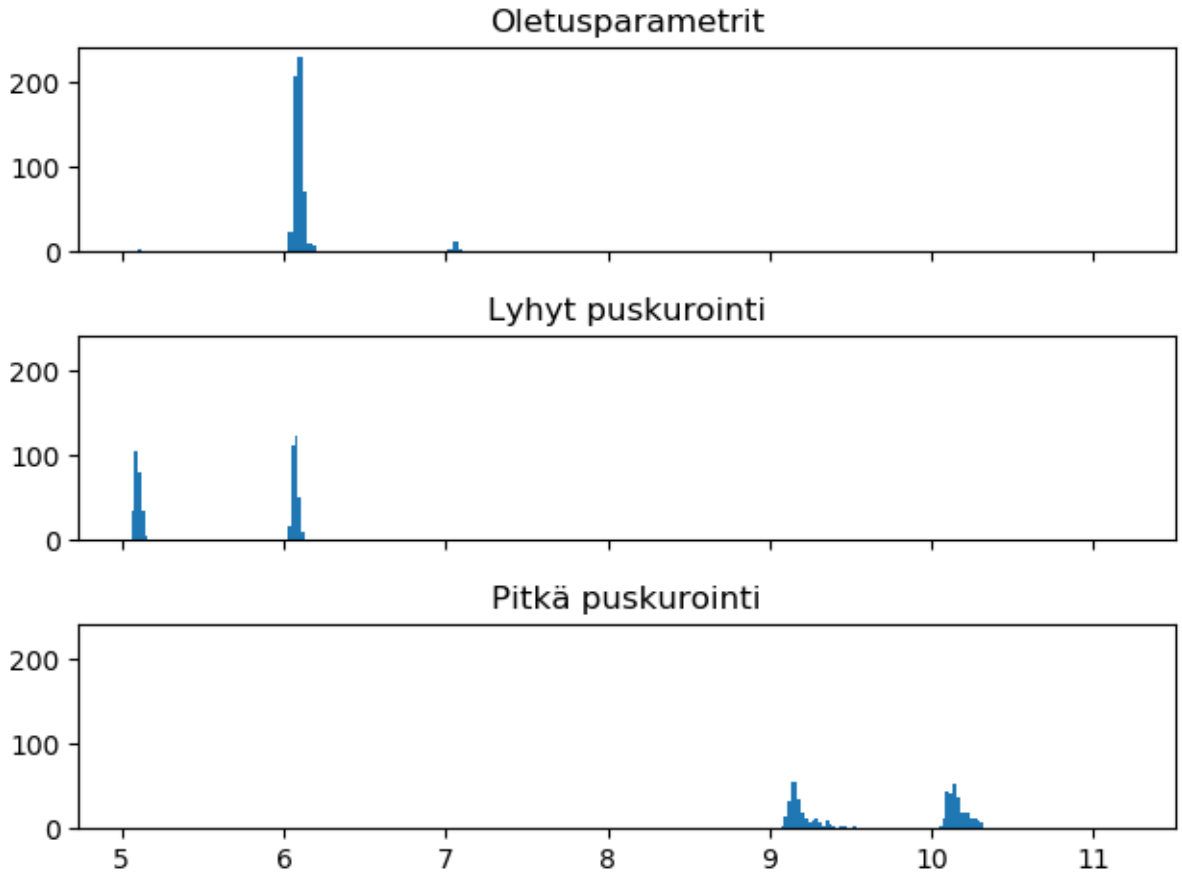


Kuva 5.3 *Puskurin pituuden muuttamisen vaikutus latensiin*

alueelle, joka on havaittavissa histogrammin pylväiden koosta. Vastaavasti herkemmäksi säädetyllä hystereesillä. Samalla on huomattavaa, että poikkeavan suuret latenssit ovat pääasiassa yksittäisiä pisteitä, sillä niiden vastaavat osuudet histogrammissa ovat pieniä.

Kuvaajassa 5.3 esitetään puskurin pituuden muuttamisen vaikutus latenssiin suhteessa aikaan. Kuvaajaan on piirretty testiajosta mittaukset, joissa puskurin pituutta on muutettu. Kuvaajassa 5.3 x-akselina toimii ottelun aika ja y-akselina mitattu latenssi. Kuvaajaan on piirretty oletusarvoilla olevat tulokset sekä pidennetyllä ja pienennetyllä puskurin arvoilla mitatut tulokset.

Kuvaajasta 5.3 näkee selkeästi, että puskurin pituuden vaihtaminen vaikuttaa tasoon, jonka ympärillä latenssi vaihtelee. Johdonmukaisesti puskurin pidentäminen aiheuttaa kuvaajan 5.3 mukaan latenssin normaalitason nousemista, vastaavasti puskurin pituuden pienentäminen aiheuttaa normaalitason laskemisen. Poikkeavana asiana kuvaajassa 5.3 on eriävien pisteiden lisääntyminen sekä pidemmillä, että pienemmillä puskurin arvoilla.



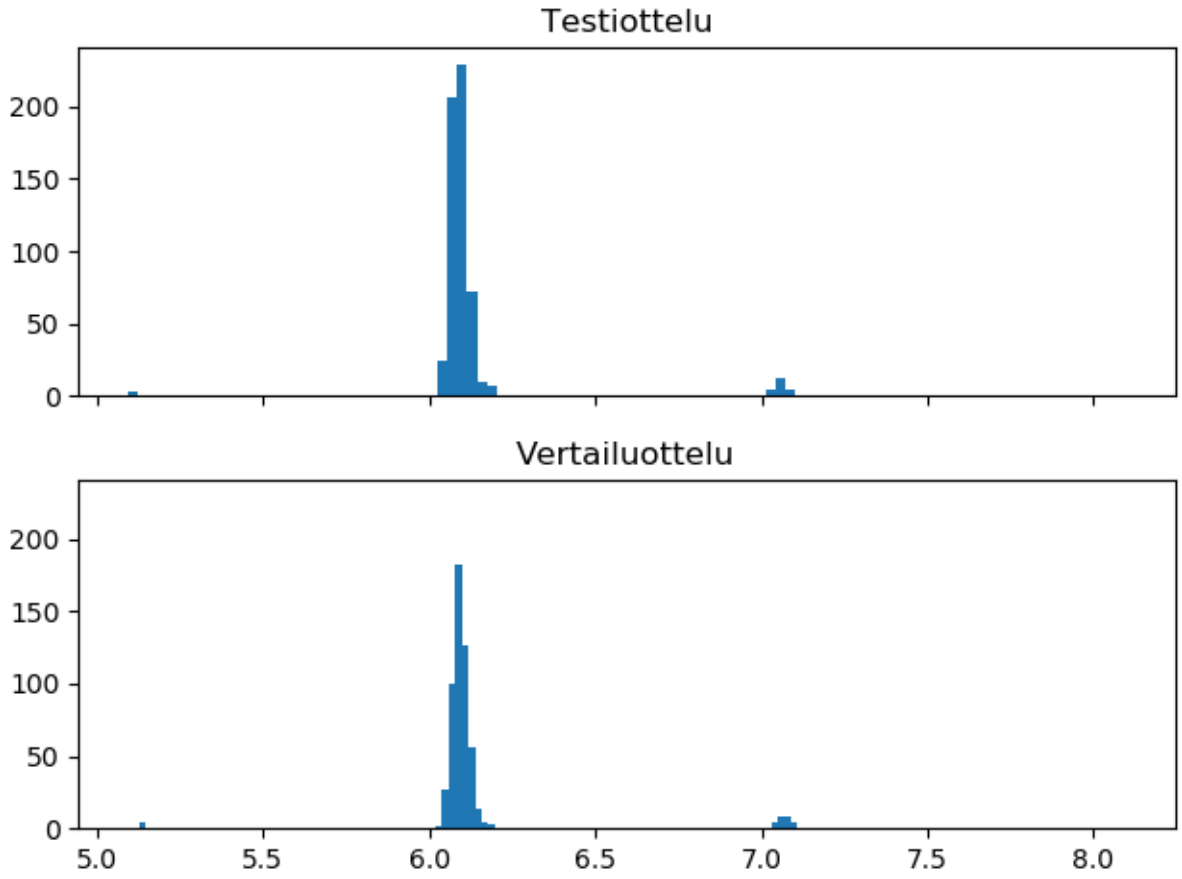
Kuva 5.4 *Puskurin pituuden vaikutus latenssiin*

Mittaus	Virheellisten tulosten määrä
Oletusparametrit	3
Pitkä pukurointi	6
Lyhyt puskurointi	3
Epäherkkä hystereesi	1
Herkkä hystereesi	4

Taulukko 5.2 *Virheellisten tulosten määrä*

Kuvaajassa 5.4 esitetään puskurin pituuden vaikutus latenssiin histogrammina. Kuvaajasta on havaittavissa, että lyhyt puskurointi aiheuttaa yleisimpi latenssin jakautuvan tasaisemmin kahteen piikkiin. Vastaava ilmiö käy pitkällä puskurointiajalla. Verrattuna pitkä puskurointi aiheuttaa levinneempiä histogrammeja verrattuna lyhyeen puskurointiin. Oletusparametreissa ei selkeää latenssin jakautumista tapahdu.

Taulukossa 5.2 esitetään virheellisten vaihtojen määrä testiottelulle eri parametrisäädöksillä. Taulukossa 5.2 esitetyistä tuloksista hystereesin herkkyyden muokkaaminen vaikutti tuloksiin oletusten mukaisesti vähentäen tarkkuutta herkkyyden kasvaessa. Oletusten vastaisesti käyttäytyy puskurin pituuden kasvattaminen, joka



Kuva 5.5 *Vertailu ottelujen latensseista*

heikentää järjestelmän tarkkuutta. Samanaikaisesti puskuroinnin lyhentäminen ei vaikuttanut tuloksiin.

Kuvaajassa 5.5 esiteään histogrammit kahdelta eri testiottelulta. Mittaukset otteluille on tehty käyttäen oletusarvoisa parametrejä. Kuvaajasta 2.1 voi havaita latenssin käyttäytyvän samankaltaisesti molemmilla testiotteluilla painottuen vahvasti yksittäiselle alueelle. Molemmilla testiotteluilla kuvaajan 5.5 mukaan oli satunnaisia poikkeamia latenssissa.

5.3 Tulosten yhteenveto

Työssä tehtiin mittauksia toteutetun ohjelmistokomponentin käyttäytymiseen muutettaessa sen parametrejä. Muutetut parametrit olivat hystereesin tilamuutoksen herkkyys ja sisäisen puskuroinnin pituus. Mittaukset parametrien vaikutuksista esiteltiin kappaleessa 5.2.

Tuloksissa esitettiin taulukossa 5.1 parametrin vaikutukset latenssiin numeerisesti.

Samalla kuvaajissa 5.2 ja 5.4 esitetään latenssiin käyttäytyminen histogrammeina vastaavilla parametrin muutoksilla. Taulukon 5.1 mukaan lyhyin latenssi saadaan lyhentämällä puskurointia. Lyhyt puskurointi aiheuttaa samanaikaisesti lyhyimmän minimi- ja maksimilatenssin. Huomattavaa on lyhyen puskuroinnin vaikutus latenssin varianssiin, joka nousee merkittävästi. Latenssin jakautuneisuus lyhyellä puskuroinnilla on havaittavissa myös kuvaajasta 5.4, jossa selkeästi tulokset jakautuvat kahteen pääkeskittymään. Jakautuminen itsesään ei ole haitallinen tekijä, sillä järjestelmä toimii latenssin kannalta keskimääräisesti paremmin kuin oletusarvoillaan. Vastakkaisesti pidempi puskurointi tuottaa hitaamman vasteajan heikentäen järjestelmän toimintaa.

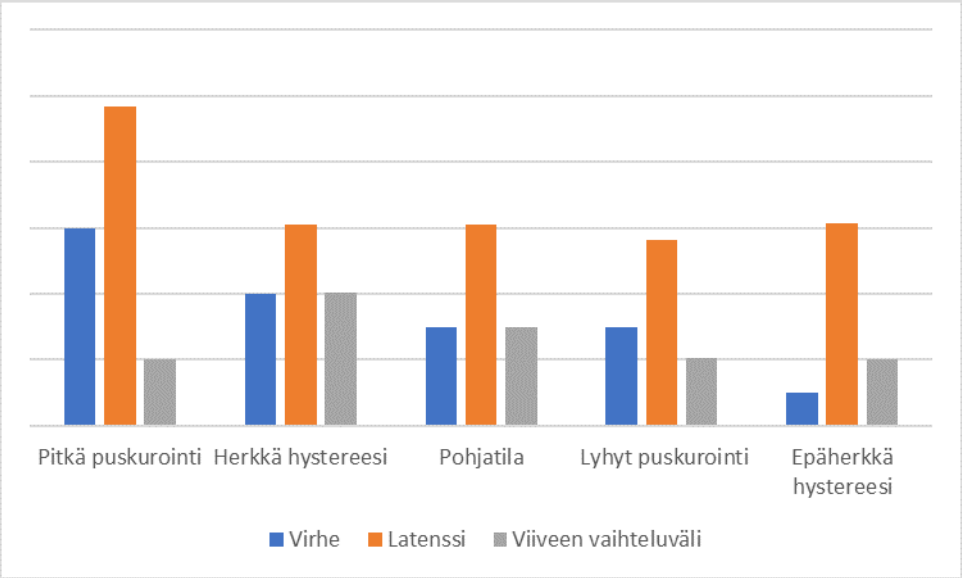
Hystereesin herkkyyden vaihtamisella ei latenssin käyttäytymiseen ole huomattavaa merkitystä. Epäherkempi hystereesi laski maksimilatenssia, mutta ei vaikuttanut minimiin tai keskiarvoon. Vastaavasti herkkä hystereesi aiheuttaa maksimilatenssin kasvamisen, mutta ei vaikuta suuresti keskiarvoon tai minimilatenssiin. Muutokselisesti hystereesin herkkyys vaikuttaa latenssin jakautuneisuuteen, joka on havaittavissa kuvaajassa 5.2. Toiminnallisuuden kannalta hystereesin herkkyyden laskeminen tuottaa paremman tuloksen latenssimittausten perusteella.

Taulukossa 5.2 esiteltiin parametrin muutosten vaikutus virheellisiin tunnistuksiin. Epäherkällä hystereesillä tarkkuus komponentin tarkkuus parani selkeästi ja puskuroinnin lyhentämisellä ei ole vaikutusta. Taulukon 5.2 tuloksien perusteella järjestelmä toimii parhaiten kun hystereesin herkkyyttä lasketaan.

Taulukkojen 5.1 ja 5.2 tuloksien perusteella järjestelmän toiminta paranee, jos puskurointia vähennetään tai hystereesin herkkyyttä lasketaan. Mittausten perustella kummassakin tapauksessa komponentin toiminta paranee näkyen, joko viiveen laskemisena tai tulosten tarkkuuden paranemisena. Sillä mittauksissa ei tutkittu parametrien yhtäaikaisia muutoksia, on todennäköistä, että tulokset paranevat, jos puskurointia vähennetään samanaikaisesti kuin lasketaan hystereesin herkkyyttä.

Kuvaajassa 5.6 esitetään taulukkojen 5.1 ja 5.2 tulokset yhteenvedettynä. Kuvaajassa 5.6 esitetään kiteytettynä mittaustulokset ja kuvaaja tukee yllä tehtyä johtopäätelmää parametrien vaikutuksesta. Kuvaajassa esitetään tuloksen virheiden kannalta laskevassa järjestyksessä. Samalla kuvaajassa esitetään latenssin vaihteluväli.

Esitetyistä viiveen mittaustuloksissa on virhettä, joka johtuu esimerkiksi vasteajan mittauksesta johtuvasta ylimääräisestä kuormasta tilastointijärjestelmään. Samanaikaisesti kokonaisjärjestelmän rinnakkaisuudesta johtuen ohjelmiston suoritusjärjestys ei ole taattu, jolloin mitattu latenssi vaihtelee testiajojen välillä. Samalla viiveen mittauksissa virheeseen vaikuttaa testiajossa käytetyn datavirran syöttö-



Kuva 5.6 *Yhteenveto olennaisista tuloksista*

nopeus. Datavirtaa hidastettiin testiajossa mallintamaan todellista käyttötapausta, kuitenkaan hidastus ei mallinna virheettä oikeaa syöttönopeutta vaan dataa syötetään tilastointijärjestelmään purskeissa, joiden välissä syöttöä tauotetaan. Purskemainen syöttäminen saattaa aiheuttaa kuvaajissa 5.3 ja 5.1 havaittua latenssin vaihtelua.

Tulosten perusteella voidaan todeta, että toteutettu ohjelmistokomponentti täyttää asetetut vaatimukset. Havaitut virheet ovat siedettävissä rajoissa ja latenssi ei aiheuta merkittävää ongelmaa tilastointijärjestelmän toiminnalle. Vaihtojen laskentakomponentin toiminta kattaa tilastoinnille esitetyt vaatimukset ja tukee muiden tilastointijärjestelmän komponenttien toimintaa.

5.4 Jatkokehitysehdotuksia

Työssä toteutettiin ohjelmistokomponentti, joka laskee pelaajien vaihdot käyttäen paikannusdataa. Toteutuksille ja mittauksille on selkeitä jatkokehityskohteita. Yleisesti järjestelmän suorituskykyä voi parantaa vapauttaen prosessoriaikaa muille tilastoinnin komponenteille.

Nykyisellä toteutuksella selkeä jatkotutkimuskohde olisi toteuttaa hienojakoisempia mittauksia parametrien vaikutuksista. Samoissa mittauksissa pitäisi validoida tuloksien toistuvuus useammalla ottelulla, sillä työssä mittauksia toteutettiin vain kahdella ottelulla. Mikäli mittauksien tulokset olisivat systemaattisesti samankaltaisia usealla testiottelulla, voitaisiin järjestelmän parametrejä optimoida tuloksien perusteella.

Mittauksia järjestelmän aiheuttamiin virheisiin voidaan toteuttaa annotoimalla ottelu videopohjaisesti. Videopohjaisella tarkisteella saataisiin määritettyä todellisuutta paremmin vastaava haluttu lopputulos järjestelmälle. Haluttua tulosta vastaan ohjelmistokomponentin jatkokehitystä voidaan toteuttaa ja vertailla.

Ohjelmistokomponentin toteutukseen voitaisiin myös kokeilla koneoppimista, sillä mallin monimutkaistuessa toteutuksen ylläpitäminen muuttuu hankalammaksi. S Koneoppimissa malli koittaisi luokitella onko pelaaja jäällä vai vaihtoaitossa käyttäen syötteena annotoituja dataa pelaajan jäälläolosta. Tällöin koneoppimismalli olisi binäärinen luokitteluongelma aikasarjasta paikannuksia pelaajan jäälläoloon.

Koneoppimispohjainen malli saattaisi tuotta tarkempia tuloksia, sillä ihmisen käyttäytymisen ohjelmoiminen perinteisin keinoin on haastavaa. Samalla koneoppimispohjainen ratkaisu saattaisi toteuttaa matalamman latenssin, mikäli puskuroinnin tarve vähenisi. Todennäköisesti paras tulos tulisi käyttäen yhdistäen mallin tuottamiin tuloksiin muuta logiikkaa, sillä osa mahdollisista virheellisistä luokitteluista, on mahdollista käsitellä ja korjata ohjelmalisesti.

6. YHTEENVETO

Diplomityössä oli tavoitteena toteuttaa ohjelmistokomponentti, joka jääkiekko ottelusta satavan paikkatiedon perusteella laskee pelaajien vaihdot. Toteutettu komponentti integroitiin osaksi laajempaa tilastointijärjestelmää. Työssä myös mitattiin numeerisesti toteutetuksen toimintaa ja arviotiin karkeasti ohjelmistokomponentin parametrien muutamisen vaikutusta toteutetun ohjelmistokomponentin toimintaan.

Toteutukselle asetettiin rajoitteita sekä työn sisäisesti, vaatien latenssin minimointia ja vikasietoisuutta toteutukselta, sekä ulkoisesti rajoittaen käytettyjä toteutusteknologoita. Toimiakseen toteutuettu ohjelmisto komponentti vaihtojen seurantaan joudutaan kasvattamaan järjestelmän vasteaikaa kompensoidakseen mittausvirhettä ja inhimillisestä toiminnasta johtuvia vikatiloja. Järjestelmän latenssi ei johdu kokonaan toteutetusta komponentistä, vaan integraation kohdejärjestelmässä on muista toteutuksista johtuvaa viivettä.

Työssä käydään läpi mittauksia järjestelmän vasteaikaan ja selkeisiin virhetapauksiin. Mittaukset keskittyivät kartoittamaan ohjelmistokomponentin parametrien vaikutusta mitattuihin arvoihin. Mittausten perusteella järjestelmän suoritusta voidaan kehittää parametrien arvojen muuttamisella. Työn tulosten perusteella ohjelmistokomponentin toteutus täyttää sille asetetut vaatimukset vikasietoisuudessa ja viiveessä.

Toteutettujen mittauksen perustella jatkokehitys- ja jatkotutkimuskohteita ovat parametrien hienosäätämisen vaikutus ja visuaalinen tulosten validointi perustuen esimerkiksi videokuvaan ottelusta.Samanaikaisesti komponentin toteutuksen voitaisiin kokeilla koneoppimista, johtuen nykyisen mallin monimutkaisuudesta ja ylläpitämisen monimutkaistumisesta. Videokuvasta luodun aineston perustella voitaisiin sekä nyisen järjestelmän tuloksia mitata tarkasti ja kouluttaa koneoppimismalli.

Työssä esitettyjen tulosten perusteella ei ole mahdollista tunnistaa yksittäisiä testiotteluja tai ottelussa pelanneita pelaajia. Työssä tehdyt mittaukset eivät täysin mallinna todellista tilannetta, sillä keinotekoisesti hidastettu syöttäminen kuormittaa testin suoritusta ja rinnakkaisen ohjelmna vuoronnus saataa aiheutaa virhet-

tä mittauksiin. Samalla mittauksissa virhettä aiheutuu myös mittausjärjestelyistä, kunten latenssin aikaleimojen talteenottamisesta.

Yhteenvetäen työssä toteutettiin ohjelmistokomponentti, joka laskee jääkiekko-ottelusta pelaajien vaihdot käyttäen paikannustietoa. Toteutus integroitiin osaksi laajempaa tilastointijärjestelmää, jossa sen suoritusta mitattiin. Mittauksien perusteella toteutus täyttää komponentille asetetut vaatimukset vasteajassa ja vikasietoisuudessa.

LÄHTEET

[1] G. Aroraa, *Building Microservices with. NET Core 2.0: Transitioning monolithic architectures using microservices with. NET Core 2.0 using C# 7.0.* Packt Publishing Ltd, 2017.

[2] A. D. Birrell and B. J. Nelson, "Implementing remote procedure calls," *ACM Transactions on Computer Systems (TOCS)*, vol. 2, no. 1, 1984.

[3] C. Bisdikian *et al.*, "An overview of the bluetooth wireless technology," *IEEE Commun Mag*, vol. 39, no. 12, pp. 86–94, 2001.

[4] "Enhancing bluetooth location services with direction finding," verkkosivu, Bluetooth SIG. [Online]. Available: https://www.bluetooth.com/bluetooth-resources/paper-enhancing-bluetooth

[5] R. Faragher and R. Harle, "An analysis of the accuracy of bluetooth low energy for indoor positioning applications," in *Proceedings of the 27th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2014)*, vol. 812, 2014.

[6] I. I. H. Federation. (2018) Jääkiekon virallinen sääntökirja 2014-2018. [Online]. Available: http://www.jaakiekkotuomarit.com/tiedostot/files/Saantokirja2014-2018.pdf

[7] D. Garlan and M. Shaw, "An introduction to software architecture," in *Advances in software engineering and knowledge engineering*. World Scientific, 1993, pp. 1–39.

[8] C. George, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*. Addison-Wesley, 2012.

[9] "grpc guides," gRPC verkkosivu, Google. [Online]. Available: https://grpc.io/docs/guides/

[10] S. Goswami, *Indoor Location Technologies*. Springer, 2013.

[11] I. Jami, M. Ali, and R. F. Ormondroyd, "Comparison of methods of locating and tracking cellular mobiles," in *IEE Colloquium on Novel Methods of Location and Tracking of Cellular Mobiles and Their System Applications (Ref. No. 1999/046)*, 1999, pp. 1/1–1/6.

[12] P. C. Jorgensen, *Software testing: a craftsman's approach*. Auerbach Publications, 2013.

[13] J.-S. Lee, Y.-W. Su, and C.-C. Shen, "A comparative study of wireless protocols: Bluetooth, uwb, zigbee, and wi-fi," in *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*. Ieee, 2007, pp. 46–51.

[14] P. McDermott-Wells, "What is bluetooth?" *IEEE Potentials*, vol. 23, no. 5, pp. 33–35, Dec 2005.

[15] A. Meister, S. Buechner, and A. Amthor, "State machine based nonlinear hysteresis model," *Mechatronics*, vol. 31, pp. 215 – 221, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957415815001257

[16] "Common language runtime (clr) overview," verkkosivu, Microsoft. [Online]. Available: https://grpc.io/docs/guides/

[17] "Dataflow (task parallel library)," Dotnet-foundation verkkosivu, Microsoft. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/index

[18] "Task-based asynchronous programming," Dotnet-foundation verkkosivu, Microsoft. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/task-based-asynchronous-programming

[19] Mircosoft. (2018) C# 6.0 language specification. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/index

[20] S. Newman, *Building Microservices*. O'Reilly, 2015.

[21] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *ACM SIGSOFT Software engineering notes*, vol. 17, no. 4, pp. 40–52, 1992.

[22] T. Scheibler, F. Leymann, and D. Roller, "Executing Pipes-and-Filters with workflows," in *2010 Fifth International Conference on Internet and Web Applications and Services*, May 2010, pp. 143–148.

[23] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Computing Surveys (CSUR)*, vol. 22, no. 4, pp. 299–319, 1990.

[24] "Global positioning system standard positioning service performance standard," verkkosivu, US Department of defence. [Online]. Available: https://www.gps.gov/technical/ps/2008-SPS-performance-standard.pdf